PATENT ABSTRACTS OF JAPAN

(11)Publication number : 05-261983

(43)Date of publication of application : 12.10.1993

(51)Int.CI. B41J 5/44
B41J 2/485
G06F 3/12
G06F 15/16
G09G 5/22

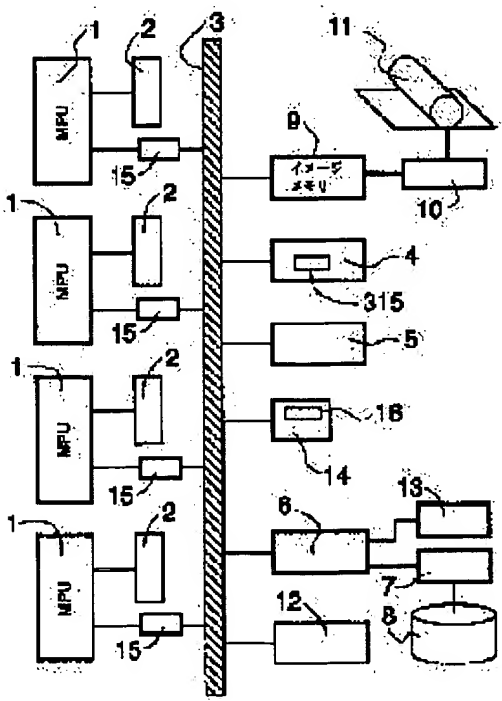
(21)Application number : 04-062395

(22)Date of filing : 18.03.1992

(71)Applicant : SEIKO EPSON CORP

(72)Inventor : NAGASAKA FUMIO

(54) CHARACTER DATA CONTROL DEVICE



(57)Abstract:

PURPOSE: To provide a character data control device of common memory type tight coupling multi-microprocessor system wherein contents of font caches in processors are made to be the same and excessive differences in processing speed among the processors are not generated.

CONSTITUTION: Font caches are arranged in local memories 2 of a plurality of microprocessors 1 and a common RAM 4 is provided with a common font cache 315. When a character data acquisition request generates, at first a reading from the local font caches 2 is attempted. When it is impossible, secondly a reading from the common font cache 315 is attempted. Further, when it is impossible either, character is generated from an outline font source data and a generated character-shaped image data is written only in the common font cache 315. The local font caches 2 are renewed at switching of process.

LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Machine English translation of JP 05-261983A

CLAIMS

[Claim(s)]

[Claim 1] It has an outline font feed zone accessible in common and a partial font cache accessible to said processor each from a share font cache accessible in common from two or more processors and these processors, and said processor, and reading from a local font cache is first tried to the acquisition demand of the alphabetic character at the time of a processes run, and if it is, this will be read, and if there is nothing, when reading from a share font cache will be tried and there will be nothing further, former data are read from an outline font feed zone, alphabetic character pixel data are generated, and it registers with a share font cache. Alphabetic character data control equipment characterized by transmitting the contents of the share font cache to a local font cache, and updating the contents of each font cache in the process termination at the time of activation, or relaxation time.

DETAILED DESCRIPTION

[Detailed Description of the Invention]

[0001]

[Industrial Application] This invention relates to the alphabetic character data control equipment which performs parallel processing by two or more processors, and outputs printing data.

[0002]

[Description of the Prior Art] In the airline printer used as peripheral devices, such as a workstation, the outline font which described the character outline independent of the resolution of an output unit as curvilinear information has been used from the font of the bit map form depending on the resolution of an output unit recently. However, in order to perform processing changed into the pixel data which finally suited resolution with each output unit, the conversion will take time amount. To the font which has a complicated profile like the kanji especially, it is still more so.

[0003] Then, about the alphabetic character [finishing / conversion] already processed by pixel data by the processor, the pixel data is temporarily stored in the memory called the font cache where it connects with the processor through the bus, and when the output request of the alphabetic character occurs again, the approach of using the pixel data is taken. Thereby, although the processing time in the case of printing can be shortened, it becomes important how only the alphabetic character with specific high operating frequency is held from there being a limitation in the capacity which it can have as a font cache. JP,64-88660,A and JP,2-233269,A are indicated as a management method of this font cache. Preventing un-arranging [which the

former resets registration ranking when the alphabetic data registered into the font cache is called again, and is deleted as old data of a registration stage], the latter gives deletion priority information beforehand to the font used.
[0004] However, even if it uses a font cache for max efficiently, with the equipment in a uniprocessor, a limitation will be too generated in improvement in the speed.
[0005] Then, the bus connection of two or more processors is carried out, and the way of thinking in which you are going to make it perform high-speed processing unrealizable by the uniprocessor is produced by the parallel processing by two or more processors. According to the computer system, there is what performs parallel processing by two or more processors, and each processor has already realized improvement in the speed of instruction access and a data access in activation of the assigned process by transmitting a part of run unit under current processing to cache memory.
[0006]
[Problem(s) to be Solved by the Invention] To it, the processing changed into alphabetic data from an outline font as mentioned above is needed, and it does not go to the reason for saying that what is necessary is to program an activation process to **** like the former computer system, and just to transmit to cache memory by the airline printer, but the time amount spent on the writing to a font cache becomes large far with it. Therefore, it is necessary not to, perform transform processing from an outline font to alphabetic data if possible, but to fill the contents of the font cache with an airline printer by data with high operating frequency moreover.
[0007] Moreover, if each processor performs font cache management uniquely respectively and this alphabetic character is not registered into the font cache of another processor even if it is the alphabetic character already changed into alphabetic character pixel data by a certain processor, duplication processing which performs conversion to alphabetic character pixel data again is performed, and it is not efficient. When an activation process with many alphabetic character kinds used is especially assigned till then to the processor which was low as for operating frequency suddenly, the utilization factor of a font cache will fall remarkably and will become a bottleneck on processing.
[0008] This invention is made in view of such a problem, and the place made into the purpose is to offer the alphabetic character data control equipment which raises nothing and the use effectiveness of a font cache for the contents of the font cache which each processor has as it is the same, and performs a high-speed document processing system.
[0009]
[Means for Solving the Problem] The share font cache where the alphabetic character data control equipment of this invention is accessible in common from two or more processors and these processors, An outline font feed zone accessible in common from said processor, It has an accessible partial font cache in said processor each. Reading from a local font cache is first tried to the acquisition demand of the alphabetic character at the time of a processes run. If it is, this will be read, and if there is nothing, when reading from a share font cache will be tried and there will be nothing further, former data are read from an outline font feed zone, alphabetic character pixel data are generated, and it registers with a share font cache.
[0010] The contents of the share font cache are transmitted to a local font cache, and it is characterized by updating the contents of each font cache at the process termination at the time of activation, or relaxation time.
[0011]
[Function] According to this invention, the writing of new alphabetic character pixel data is

performed to a share cache. Since data are transmitted to each local cache from a share cache at relaxation time, process termination **** can guarantee the identity of a local cache.
[0012]
[Example] Drawing 1 is the block block diagram of the airline printer of the electrophotography method held as one example of this invention.
[0013] An airline printer consists of three main parts. That is, they are the print engine 11, an engine control system 10, and a control circuit. The actuation of an airline printer is as follows. A printing control code is received from the interface 13 by which the control circuit was connected to devices, such as an external personal computer. The control processing program performed by the microprocessor unit 1 interprets this, generates a printing pixel, and writes in image memory 9. An engine control system 10 transmits this pixel data to an engine side synchronizing with actuation of the print engine 11, and performs printing processing.
[0014] This example takes in the means of this invention about the alphabetic character data control method of a control circuit among a series of above-mentioned processings.
[0015] The configuration of a control circuit is as follows. The microprocessor unit 1 of plurality (drawing 1 four pieces) is connected to the global bus 3 by the bus mediation circuit 15. The bus mediation circuit 15 arbitrates contention of the microprocessor unit 1 to the global bus 3 according to the control signal of the bus mediation control circuit 12. Share ROM 5 and share RAM 4 are connected to the global bus 3. The control program of the airline printer of this example is written in the share ROM 5. Moreover, share RAM 4 is used for the working-level month memory of the above-mentioned control program, and the global font cache and receive data buffer which are mentioned later.
[0016] In the airline printer of this example, bit map printing, alphabetic printing, and easy graphic form printing are possible by assignment of a control code. However, in explanation of this example, since it is easy, only the case of alphabetic printing is explained.
[0017] The airline printer of this example performs alphabetic printing using an outline font. The former data of the outline font to be used are beforehand recorded on the hard disk drive unit 8. A hard disk drive unit 8 is controlled by I/O processor 6 through a disk controller 7. Moreover, the interface 13 of an external computer capital is also controlled by I/O processor 6. Since I/O processor 6 is accessed by the communication link port from the global bus 3, each microprocessor unit 1 is uniquely accessible to I/O processor 6.
[0018] The printing pixel which each microprocessor unit 1 generates is recorded on image memory 9. Image memory 9 is constituted by dual port memory, and it is timing different from mediation of the global bus 3, and the engine control circuit 10 reads pixel data, and it transmits it to the print engine 11.
[0019] On the other hand, the respectively local font cache 2 is connected to each microprocessor unit 1. Local bus connection of this font cache 2 is made, and as compared with the memory accessed through the global bus 3, since bus mediation is not needed, high-speed access can be performed.
[0020] Next, actuation of the font cache management method used by this example is explained using drawing 2 . In the font cache 2 locally connected to each microprocessor unit 1, the font managed table 206 and the font data section 201 are arranged. Moreover, in the global font cache field 315 placed after share RAM 4, the font managed table 206, the font data section 201, and an overflow area 205 are arranged. For the font managed table 206, in this example, 6 K bytes and the font data section 201 are [128 K bytes and an overflow area 205] 2 K bytes.
[0021] The DS on each [these] memory is the same also in the font cache 315 on share RAM 4

also in the local font cache 2. In drawing 2 , since it was easy, it considered as the explanatory view which accesses these DS with an address bus 215 and a data bus 216. In these actual bus access, local memory access is performed by the local bus of a microprocessor unit 1, and global access is performed by the global bus 3.
[0022] Such DS is explained separately below.
[0023] The font managed table 206 is an one-dimensional array which consists of 300 of the structure constituted by the alphabetic character assignment record 208 of 207 or 12 bytes of registration flag [6 bytes of], and 2 bytes of key 209. This structure was shown in drawing 6 . Here, an alphabetic character assignment record is DS which consists of the following elements.
[0024]

However, a typeface is a name used when a Hollerith type-like difference is specified, when two or more alphabetic character designs are performed about a certain alphabetic character. It is used in the above-mentioned DS, assigning a numeric value to this name. Moreover, character decoration assignment is not based on an alphabetic character typeface, but it is the assignment which makes the configuration of the target alphabetic character applicable [, such as an italic character and a void alphabetic character,] to actuation, and the internal representation in the case of using it by the above-mentioned DS is evaluated like the typeface.
[0025] The font data section 201 is memory with address arrangement [**** / finite length]. The interior is the set of the management unit memory 202 with a magnitude of 128 bytes. The management unit memory 202 consists of data division 204 with a magnitude of 126 bytes and an index part 203 with a magnitude of 2 bytes. The contents of the index part 203 are the indexes to the management unit memory which should follow a degree. Since the management unit memory 202 is magnitude immobilization, it can calculate the value of the real address by the easy operation from the value of an index. That is, when the value of an index is set to ix, the head address of the management unit memory 202 which should follow a degree is obtained as a value addr of a degree type.
[0026]
addr = ix x 128 + There is nothing to memory 201 head address ****, and a value 128 comes from the magnitude of management unit memory, and this operation is only an example of conversion to the real address. The index part 203 of the management unit memory 202 is one of the deformation of the so-called connection pointer.
[0027] the image data of an actual alphabetic character -- the data division 204 of the management unit memory 202 -- secondary -- it is used some and stored. The contents of the data stored are the number of the management unit memory 202 which 2 bytes of the beginning use, and the data following this are an image data. Moreover, all image datas consist of a data stream by which run length compression was carried out.
[0028] There is the approach of arranging a variable-length data stream dynamically on the continuous memory as the memory management technique as everyone knows. If this approach is used, after repeating the registration and deletion to a cache, the free area which is depended on the data stream which became unnecessary and which two or more magnitude does not have will arise. In order to rearrange such a fragmented memory resource that became unnecessary and to make the continuous big free area again, there is the need of performing processing which the execution time including the block move of memory requires. On the other hand, an intact part arises in the management unit memory of the last edge, and a means to use two or more

fixed-length management unit memory which this example adopted is inferior to dynamic memory management in respect of a memory utilization factor. However, from management being simple,-izing can be carried out [hardware] easily, and also even if it performs a manager by software, it excels in the point which can perform high-speed processing.
[0029] An overflow area 205 is an one-dimensional array with a continuous magnitude of 2 K bytes.
[0030] The above font data sections 201, the font managed table 206, and an overflow area 205 are managed by a series of processing groups which each calls the cache quota processing 217. The read-out processing 210, the write-in processing 211, and the connection dissolution processing 212 are included in these processing group.
[0031] Next, the cache quota processing 217 is explained.
[0032] About the alphabetic character registered into the cache, the image data is divided and stored in two or more management unit memory 202 at the head in the management unit memory 202 of one of the font data sections 201. About the management unit memory 202 which comes first at this time, 300 management unit memory 202 is continuously used from the head of the font data section 201, respectively. Management unit memory 202 index of the beginning in the case of recording the data for a single character on two or more management unit memory 202 is specified by the key 209 in the structure of the font managed table 206. Although the font data section 201 has the magnitude of about 128 K bytes in the appearance described previously, this is 1024 sets of the management unit memory 202. Since there is a limit of such a size of an array, in this example, the image data per character is about an average of 430 bytes.
[0033] Next, the flow of processing of alphabetic character data acquisition is explained using drawing 5 .
[0034] When the demand which acquires alphabetic data occurs within a certain process, the alphabetic character data acquisition processing 501 is called. A demand is performed by alphabetic character assignment record which was mentioned above here. It is because it is necessary to treat the alphabetic character assignment from which magnitude, an alphabetic character kind, and the character decoration differ needless to say even if it is the same character code.
[0035] The procedure of the alphabetic character data acquisition processing 501 is as the flow chart of drawing 8 . The microprocessor unit 1 while performing processing searches the local font managed table 206 first (S801). If the same alphabetic character assignment record as the alphabetic character assignment record given here exists in the font managed table 206, alphabetic data (Hollerith type-like image data) exists in a local font cache. Therefore, read-out from a local font cache is performed (S802).
[0036] However, when the data of the alphabetic character which corresponds by retrieval of the local font managed table 206 are not able to be detected, the global font managed table 206 is searched continuously (S803). When the same alphabetic character assignment record as an alphabetic character assignment record exists in the font managed table 206, the value of the access counter 508 which is a global share variable is carried out +one (S804). An access counter 508 is a counter of 48 bit length, and is cleared at the time of a system startup. It is not cleared henceforth, but when there is an element newly registered into the font managed table 206 by the alphabetic character data acquisition processing 501 again when the candidate for retrieval was detected by the font managed table 206, +one is taken by the alphabetic data registration processing 502, respectively. Overflow is not generated in the busy condition of this example. The value of an access counter 508 is recorded on the registration flag 207 corresponding to the

element with which the shared font managed table 206 was detected (S805). It can be judged that the value of an access counter 508 is the element accessed recently, so that this number is large, in order to count the count to which registration and read-out were performed.
[0037] Then, the key 209 of the element with which the font managed table 206 corresponds is taken out (S806), and alphabetic data read-out from a global font cache is performed (S807). On the global font managed table 206, when the element corresponding [an alphabetic character assignment record's] is not able to be detected, reading from the hard disk drive unit 8 of alphabetic data is required from a kernel (S808).
[0038] This is accompanied by the I/O process. In this example, between I/O processes, each microprocessor unit 1 does not wait for a processing result, but stops the process under current activation (S809). At this time, the update process 304 shown by drawing 3 is performed. Update process 304 locks exclusively access of other microprocessor units 1 to a font cache field shared [RAM / 4] first. Next, all of the global font managed table 206 and the contents of the font data section 201 are transmitted to the local font managed table 206 and the font data section 201. In this example, this processing time is about 40m second. The exclusive control of access to the font cache field on share RAM 4 is canceled after this processing termination.
[0039] On the other hand, termination of reading of the outline font data from a hard disk drive unit 8 performs I/O termination processing by I/O processor 6. First, the flag which shows the kernel program actuation on a shared memory is checked, and there is a microprocessor unit while performing a kernel program, or detection is performed.
[0040] When the microprocessor unit 1 while performing a kernel program exists, I/O processor 6 carries out the I/O termination flag on a shared memory truly, and completes an I/O process. When the microprocessor unit 1 while executing a kernel program does not exist, I/O processor 6 sends an interruput signal to the waiting microprocessor unit 1 after carrying out the I/O termination flag on a shared memory truly using the interrupt control circuit 14. Here, the interrupt control circuit 14 of one waiting microprocessor unit waits to suspend processing and to stop activation of one of processes, when there is nothing.
[0041] When a kernel program is executed and termination of processing of I/O processor 6 is detected, process starting is urged to a kernel program by interruption to the waiting microprocessor unit 1, and the re-start process of a process stopped previously is performed (S810). However, the resumed process is not necessarily performed by the same microprocessor unit 1 as a pause or before.
[0042] The resumed process acquires the alphabetic data of an outline font by kernel service (S811), and generating processing of the image data showing the configuration of an alphabetic character is performed (S812). Although this processing is based on Hollerith type-like complexity, it is a kanji with a magnitude of 40x40 pixels, and this example takes the about [200m second] processing time from an average of 100m second. Then, the microprocessor unit 1 while performing a process registers the image of the generated alphabetic character only into a global font cache by the call of the alphabetic data registration processing 502 (S813).
[0043] In this example, the local font cache 2 does not write in except update process 304 as mentioned above. The local font cache 2 is always only set as the object of read-out.
[0044] Drawing 9 is the flow chart of processing of the alphabetic data registration processing 502.
[0045] From the alphabetic character data acquisition processing 501, the alphabetic data registration processing 502 is called as an argument, and the image data and the alphabetic character assignment record 208 of an alphabetic character are used. The registration processing

502 calls the table retrieval processing 213 (S901), searches the registration flag 207 of the font managed table 206, and takes out the smallest element of a value (S902). If the value of the registration flag 207 is 0 at this time, it will be judged that that element is an element which was not set as the object of font registration until now, or was just deleted. Then, the contents of the registration flag 207 are rewritten to FFFFFFFFFFFF of a hexadecimal. This is a temporary flag which shows that it is [current] under use. Then, the contents of the key 209 corresponding to this element are taken out, and write-in processing 211 is performed for the image data of a key and an alphabetic character to an argument (S903). As a return value of a processing result, when an overflow area error is returned, nothing is performed but registration processing is ended. In this case, at the time of registration processing termination, the value of the registration flag 207 is restored to the value before processing.

[0046] As a return value of the write-in processing 211, when a write error is returned, the registration processing 502 calls deletion 503 (S904). Since a free area arises in the font data section 201 of a font cache system as a result of this processing, the registration processing 502 calls the write-in processing 211 by making a key and an image data into an argument again. Furthermore, as a return value of a processing result, when a write error is returned, the same processing is repeated and registration is completed. In this case, +1 is carried out to the contents of the access counter 508 at the time of registration processing termination (S905), and this value is written in the registration flag 207 of the shared font managed table 206 (S906).

[0047] If this value is not 0 when the registration flag 207 of the font managed table 206 is searched and the smallest element of a value is taken out (S902), deletion 503 will be called (S907), a free area will be secured, and processing after S902 will be performed.

[0048] Drawing 10 is the flow chart of processing of the alphabetic character data deletion processing 503.

[0049] The alphabetic character data deletion processing 503 calls the table retrieval processing 213 (S1001), and takes out the minimum element which is not 0 about the registration flag 207 of the shared font managed table 206 (S1002), and the value of the key 209 corresponding to this element is acquired (S1003). Next, the connection dissolution processing 212 is called by making the value of this key into an argument (S1004), and processing is ended. As drawing 5 showed, the processing group of the cache quota processing 217 is called from the alphabetic character data acquisition processing 501, the alphabetic data registration processing 502, and the alphabetic character data deletion processing 503.

[0050] Next, the read-out processing 210, the write-in processing 211, and the connection dissolution processing 212 are explained among the cache quota processings 217, respectively.

[0051] Processing of the read-out processing 210 is explained using the flow chart of drawing 11
.

[0052] The read-out processing 210 takes out the key received as an argument (S1101), calls the address-arithmetic section 214 (S1102), and acquires the pointer to the management unit memory 202 of the font data section 201 (S1103). Then, according to this pointer, it accesses to the management unit memory 202 used as the head of image-data storing of an alphabetic character (S1104). According to the index to the management unit memory 202 connected below, the image data of an alphabetic character is read one after another (S1105). Since the already described contents of the data stored like are the number of the management unit memory 202 which 2 bytes of the beginning use, they can connect the management unit memory 202 of the required number. Moreover, the contents of the index part 203 of the management unit memory 202 located at the last of a series of connection are -1 (it is FFFF in a hexadecimal).

[0053] Processing of the write-in processing 211 is explained using the flow chart of drawing 12 .

[0054] The write-in processing 211 takes out the value of a key from the received argument (S1201), and calls the address translation processing 214 for this value to an argument (S1202). As this processing result, the pointer to the start address of the first usable management unit memory 202 is acquired in image-data writing (S1203). Next, the write-in processing 211 accesses this management unit memory 202 (S1204), and writes in the contents of the integral value (2 bytes) which broke the total data size of an image data by "the size -2 of management unit memory." This is in agreement with the total of the management unit memory 202 required for data storage. Next, an image data until it results in the magnitude of data division 204 is written in succeeding 2 bytes of this data (S1205). Further, if there is the remaining byte count of an image data, write-in processing writes this data in an overflow area 205 first (S1206), it will be the font data section 201 interior of cache memory, and the intact management unit memory 202 will be looked for (S1207). When the intact management unit memory 202 is able to be discovered, the index to the current management unit memory 202 is written in the index part 203 of the management unit memory 202 accessed previously, and new association is generated (S1208). Then, from an overflow area 205, 126 bytes (this is the size of the data division 204 of the management unit memory 202) of data are taken out, and it writes in data division 204 (S1205). This is repeated until the contents of the overflow area 205 become empty or the intact management unit memory 202 is lost. When the remaining data of an overflow area 205 are lost, write-in processing is terminated normally. In this case, the contents of the index part 203 of the management unit memory 202 used at the end are rewritten to the value FFFF which shows joint termination, and the termination of association is generated (S1209).
[0055] When only the number which needs the intact management unit memory 202 cannot be detected on the other hand but the image data remains in the overflow area 205, the write-in processing 211 returns a write error to the processing by the side of a call (S1210), and stops processing. In this case, although it is necessary to remove the data of an unnecessary alphabetic character from the font data section 201 of font cache memory, to extend a free area, and to call the write-in processing 211 again, this processing is set and performed [judge and] at the call side (alphabetic data registration processing 502) which is high order processing.
[0056] Although an overflow area 205 is a memory area of the continuous address which has a sufficiently big capacity in the data of one character, when the image data of the alphabetic character of magnitude which has not gone into an overflow area 205 is passed, the write-in processing 211 returns an overflow area error (S1211), and stops processing. In this case, the image data of this alphabetic character is not registered into a font cache. Because, since especially the alphabetic character with a big image data has very low occurrence frequency, even if it registers with a cache, it is for not contributing to the processing speed improvement of a system.
[0057] Processing of the connection dissolution processing 212 is explained using the flow chart of drawing 13 .
[0058] The management unit memory 202 from which data were deleted, or the blank management unit memory 202 of data is distinguished from other management unit memory 202, when the value of an index part 203 is 0. The connection dissolution processing 212 acquires the key which is the value of the index from the argument to the management unit memory 202, and calls address translation processing 214 further (S1301). The address translation processing 214 changes a key value into the real address by the operation, and generates the pointer indicating

the first management unit memory 202 on which the alphabetic data used as the candidate for deletion is recorded (S1302). Hereafter, the connection dissolution processing 212 calculates the real address of the head address of the following management unit memory 202 from the index part 203 of the management unit memory 202 which this pointer shows, and the contents of the following index part 203 are replaced by 0000 one after another (S1304). As long as connection of the management unit memory 202 continues, this actuation is repeated until it results in the management unit memory 202 whose contents of the index part 203 are FFFF (in hexadecimal). Then, the contents of the index part 203 of the last management unit memory 202 of connection are rewritten from FFFF to 0000 (S1305). (in hexadecimal)

[0059] Above, explanation of the access means to the data in the font cache which this example used is finished.

[0060] Some well-known approaches exist as a font cache memory management method which a font control mechanism uses. Fundamentally, font cache memory is memory assignment of finite length, and when it is newly going to register the image data of an alphabetic character and the free space of memory cannot be secured, it updates current registration. That is, from a cache field, unnecessary data (here image data of one of alphabetic characters) are deleted, and new data are registered. It is important as a management method to leave required data and to delete unnecessary data. However, in order that there may be no ideal approach of predicting whether the demand to what kind of alphabetic data will occur in the future, unnecessary data will be decided in accordance with a certain criteria. If this decision is appropriate, the probability (hit ratio) for the data of a request character to exist in a cache field will improve, and it will lead to improvement in processing speed. The typical management method which determines unnecessary data is following four.

[0061] Approach 1: It is judged that the data registered most in early stages are unnecessary in a current data constellation. The so-called approach of First In First Out (FIFO).

[0062] Approach 2: It is judged that the longest data in a current data constellation by which period use was not carried out are unnecessary. The so-called approach of Least Recently Used (LRU).

[0063] Approach 3: It is the approach judged that the data which were rather used for the degree and were used recently are [the longest data in a current data constellation by which period use was not carried out] unnecessary.

[0064] Approach 4: In a current data constellation, it is judged that data with the lowest operating frequency are unnecessary. The so-called approach of Least Frequently Used (LFU).

[0065] JP,64-88660,A" font cache control-system" uses the approach which has improved the approach 3 of facing an approach 1 among the conventional approaches. Moreover, JP,2-202464,A" airline printer" uses the approach 1.

[0066] These managements method is the approach invented to the single font control mechanism to the configuration which uses single font cache memory. By multimicroprocessor method of close coupling like this invention (based on a bus share), in order to establish data coincidence of each font cache, management of a local font cache and the shared font cache needs to be performed separately.

[0067] The approach of Least Recently Used (LRU) was used for this example in local management of a cache.

[0068] The parallel execution of each microprocessor unit 1 is controlled by the interruput signal which the interrupt control circuit 14 generates. This is explained using the explanatory view of drawing 3 , and the state transition diagram of drawing 4 below.

[0069] Parallel execution of the program performed in the airline printer control circuit of this example is carried out by two or more microprocessor units 1. At this time, the run unit of each processor is called a process. A process consists of a process header, a program object, and a working area. The value of a program counter in case a process is started (include the restart after interruption), the value of a stack pointer, the value of each register of a processor, memory size, memory management information, and the management information of a process proper are contained in a process header. The program code which can perform the object part of a process is contained. Moreover, the working area of a process is assigned to the variable within a process, a stack, etc. At the time of activation, the stereo of a process is arranged in a main storage area. In this example, the room 300 as shown in share RAM 4 at drawing 3 is taken, and processes 303 and 308 and 309 grades are arranged here. Since a process is generated by the process generation demand to a kernel during program execution and it disappears by termination, the number of the process at the time of activation is unfixed.
[0070] Each microprocessor unit 1 which this example uses has a jump device to the specific address called some "exception-handling vectors." These are "the vector which should be jumped when interruption processing is performed", or "the vector which should be jumped after reset." All the microprocessor units 1 perform instruction word of the specific address specified by the vector after reset immediately after powering on. In the case of this example, this instruction word generates an endless loop and processing of each microprocessor unit will be in a standby condition. The field 301 arranged in the room 300 on the explanatory view of drawing 3 is a field where this endless loop is contained. This is called the standby routine 301 below.
[0071] Next, some interrupt control circuits 14 choose one microprocessor unit 1, and generate an interuput signal. Interruption is performed by specifying the jump place address at the time of interruption. The program counter of a microprocessor unit 1 is rewritten by this interruption by the value besides an endless loop. Here, the address which includes the jump instruction to the kernel field of an interruption dispatch table first is specified, and the jump to dispatch routine 302 occurs. Consequently, any one of four microprocessor units 1 starts the processing 306 of a kernel.
[0072] Actuation of the kernel processing 306 is shown in the flow chart of drawing 7 . Shortly after the processing is started by interruption, as for the kernel processing 306, the demand at the time of generating of interruption judges whether it is I/O termination processing (S701). If this decision is truth, I/O termination processing (S702) will be carried out. If it investigates whether a queue has a processing demand and there are S703 and a processing demand when decision of S701 is not truth, and when processing of S702 is completed, the contents of a demand will be processed (S704). Next, if the processing state of an I/O processor is acquired (S705) and the completed I/O process occurs, S701 or subsequent ones will be repeated. When all processing demands within a queue are processed, decision S703 becomes false, ends processing, and returns to the standby routine 301.
[0073] The microprocessor unit 1 which performs kernel processing assigns activation of the initiation process of a printing processing program once [of the beginning] after starting. The value of the starting address of a process is written in a dispatch table, and, specifically, interruption is generated in other microprocessor units 1 using the interrupt control circuit 14. The value of the bit to which this flag corresponds about the microprocessor unit 1 which has the flag 16 which shows the operating state of a microprocessor unit 1 in the interrupt control circuit 14, and is in the standby routine 301 is 1. Therefore, by the monitor of the value of this flag, the interrupt control circuit 14 specifies the waiting microprocessor unit 1, and sends an interuput

signal to this. Since the jump place address is written in the dispatch table by other microprocessor units 1 working at a kernel, the program counter of the microprocessor unit 1 into which the interruput signal was inputted here is already changed into the appointed address by them.
[0074] Consequently, if starting of the first printing processing program is performed, all actuation will be performed by the back according to the flow of processing of this printing processing program. A printing processing program generates the process which receives a printing control code from an external computer through an interface 13, and performs reception printing processing for a printing control code. By this middle, Hollerith type-like generating, rearrangement of a printing pixel, a transfer of the data completion signal to an engine control system 10, etc. are performed.
[0075] The process whose control circuit of this example is a run unit respectively independent of each processing which was described here since it is a multimicroprocessor configuration is assigned. Especially, about alphabetic character generating, whenever there is an input of a line feed code, a process is generated. On the other hand, if a series of alphabetic character generating of all to a line feed code is performed, the process of this alphabetic character generating will serve as processing termination, and will carry out the ENQ of the process termination demand to the request queue of a kernel (the case where the case where an element is added to a queue is removed in an ENQ and an element is written to be a dequeue). Since the target process will be removed and discarded from room 300 if a kernel has this demand, the number of processes is [dozens of] at most. With the explanatory view of <u>drawing 3</u> , the example of the room of a certain time amount and the condition of program execution is given. Three processes 303, 308, and 309, dispatch routine 302 and the demand generating processings 305 to a kernel, the kernel processing 306, I/O process 307, and update process 304 exist in room 300 in this case. The buffer 316 which, in addition to this, saves the field 315 of a global font cache (share) and the received data from an interface in room is arranged. Moreover, in <u>drawing 3</u> , a rightward arrow head shows transition of the execution time, and a downward arrow head shows the direction of an upper address on memory arrangement. Furthermore, the polygonal lines 311, 312, and 313 show the value change of the program counter at the time of activation of a microprocessor unit 1, respectively.
[0076] In this example, update process 304 realizes the main actuation of this invention. An example is given below and this is explained. When the microprocessor unit 1 which has already processed the process 303 under activation tends to generate a new process, a process generation demand is given to a kernel. This demand is performed by calling the demand generating processing 305. It is shown that for this reason a program counter writes the polygonal line 312 to the location of the demand processing 305 temporarily, and it changes it. A process generation demand is performed in the location of an arrow head A. The ENQ of this demand is carried out to the request queue to a kernel. Kernel processing 306 is performed by another microprocessor unit 1. The kernel processing 306 (the polygonal line 313 shows) takes out this demand, newly generates a process 308, and writes that activation starting address in the dispatch table in dispatch routine 302.
[0077] The microprocessor unit 1 while performing kernel processing 306 uses the interrupt control circuit 14 next, specifies another microprocessor unit 1 under current standby, and sends an interruput signal. The polygonal line 311 shows the locus of the program counter of the microprocessor unit 1 which performs a process 308 as a result. This microprocessor unit 1 receives interruption, jumps it to dispatch routine 302 first, and is jumped to the starting address

of a process 308.
[0078] When there is no waiting microprocessor unit 1 in the above-mentioned processing, it interrupts and generates and kernel processing assigns a processes run, when the microprocessor unit 1 which will be in a standby condition occurs.
[0079] The microprocessor unit 1 which is, on the other hand, performing processing previously shown with the polygonal line 312 generates an I/O process demand next in this example. As already explained in the flow chart of drawing 8 , the case where alphabetic data is read from a hard disk drive unit 8 etc. is accompanied by this processing. For this reason, the polygonal line 312 which is the locus of a program counter is again rewritten temporarily by the demand generating processing 305.
[0080] In this example, if all processes go into an I/O process, they will once stop continuation of processing of a process and will return to a standby condition. A kernel can assign the microprocessor unit 1 included in a standby condition to activation of another process by this. It is because an I/O process produces the waiting about a number ms in many cases, so it is advantageous to other processes for use [move / activation] being [a processor resource] more efficient rather than a certain processor continues processing within a process in the meantime.
[0081] All processes perform update process 304, when interrupting a process as mentioned above, and when a process is completed. The contents of processing of update process 304 are recording the present condition of a process in process interruption, and updating the contents of the local font cache 2 of a microprocessor unit 1. As this was already described, from the font cache 315 on a shared memory, the contents of the font managed table 206 and the font data section 201 are read, and it realizes by transmitting to the local font cache 2.
[0082] Moreover, in process termination, the contents of processing of update process 304 open wide the memory which the process used, and the following is updating the contents of the local font cache 2 of a microprocessor unit 1 similarly.
[0083] The polygonal line 312 which is the locus of a program counter shows signs that the update process 304 is performed in the location of an arrow head B. This microprocessor unit 1 stops a process after that, and returns to activation of the standby routine 301 (see S809 of drawing 8). At this time, the value of the bit applicable to this microprocessor unit of the flag 16 of the interrupt control circuit 14 is set to 1, and it records going into a standby condition.
[0084] On the other hand, after an I/O process is completed with I/O processor 6, an interruput signal is sent to the waiting microprocessor unit 1 by the interrupt control circuit 14. Consequently, the microprocessor unit 1 to which the locus of a program counter is expressed with the polygonal line 312 starts activation of the kernel processing 306. As the flow chart of drawing 7 already explained, I/O termination processing is first performed in this case. Therefore, the polygonal line 312 goes into the field of I/O process 307 temporarily. The kernel processing which took out the processing result of I/O processor 6 here resumes the process 303 stopped like the point. Here, since the microprocessor unit 1 which showed the locus with the polygonal line 313 is in a standby condition, as for the kernel processing 306, the interrupt control circuit 14 performs interruption processing to this microprocessor unit 1. The specified microprocessor unit 1 passes along dispatch routine 302 by this interruption processing, and processing is resumed from the address which stopped last time [of a process 303] (see S810 of drawing 8). This is a location shown by the arrow head C.
[0085] The above is the flow of actuation of the parallel execution and the update process 304 in this example.
[0086] Next, another example is given about update process 304.

[0087] In the above-mentioned example, the update process 304 was what transmits all all the font managed tables 206 and font data sections 201 to the font managed table 206 and the font data section 201 of the local font cache 2 from the font cache 315 on a shared memory. In each processing of font reading and registration, in case this updates the registration flag 207 of the font managed table 206 after incrementing the access counter 508 which is a share variable, it is for operating it only to the global font managed table (share) 206.
[0088] However, the processing which transmits all the contents of the font data section 201 on a shared memory to partial memory is seen from the point of processing speed, and makes difficult use of the font cache more than a certain size. The more effective transfer should be performed to mount the font cache of very big size (1 M bytes or more). In order to realize this, it is possible to change the following part as another example.
[0089] First, it changes so that renewal of a registration flag (S805) may be performed to the both sides of the font managed font managed table 206 and 206 on the partial font cache 2 on a shared memory 4 in the flow chart of the alphabetic character data acquisition processing 501 of drawing 8 . Moreover, it changes so that renewal of a registration flag (S906) may be performed in the flow chart of the alphabetic data registration processing 502 of drawing 9 to the both sides of the font managed font managed table 206 and 206 on the partial font cache 2 on a shared memory 4.
[0090] This can perform update process 304, as shown in the flow chart of drawing 14 . That is, the contents of the font managed table 206 on a shared memory and the font managed table 206 on a font cache 2 are compared separately (S1401). When the registration flag 207 of the font managed table 206 on a shared memory is newer than the registration flag of the homotopic of the font managed table 206 on a font cache 2 (a value is large), it transmits this element (207,208,209 wholly) of the font managed table 206 of a shared memory to the homotopic of the font managed table 206 on a font cache 2 (S1402). Next, the key of this location is taken out and the address of the beginning of the management unit memory 202 of the starting position of the font data in the font data section 201 is calculated (S1403). Furthermore, an over-write [section / 201 / on a shared memory 4 / font data / them / the contents of the management unit memory 202 are read one by one according to a connection list about all the data of a single character, and / homotopic / of the font data section 201 on a font cache 2] (S1404). At this time, deletion is unnecessary. Then, it judges whether it is termination of the font managed table 206 (S1405), and subsequent (S1401) processings are repeated till table termination. In this processing, the processing time of update process 304 is shortened as the contents of the font cache are stabilized and the frequency of registration of a new alphabetic character falls.
[0091] By the process control accompanied by the update process 304 which was stated above, the contents of the local font cache 2 of two or more microprocessor units 1 can be made in agreement, respectively in the airline printer of this example. On the other hand, data reading from a font cache 2 can be performed at a high speed by not generating access to shared memory space during a processes run.
[0092] That is, that this example showed performs font registration accompanied by write-in actuation to a shared memory, and reading is a font management method performed only from partial memory. In case a font is registered into a font cache, in order not to perform the writing to partial memory, either, memory address management is easy and its constraint on arrangement of partial memory and a shared memory is flexible few.
[0093] Moreover, in the configuration in the above-mentioned example, it cannot be overemphasized by transposing the engine control system 10 of drawing 1 , and the part of the

print engine 11 to a graphic controller, CRT equipment, etc., respectively that it is applicable to devices, such as a display and a workstation, as it is.
[0094]
[Effect of the Invention] When the local font cache has been arranged [in / so that clearly / the processor of a multiprocessor configuration] to each microprocessor in the above example, it is easy to maintain the data consistency in a cache with the alphabetic character data control means of this invention. In addition, this invention is a solution means to aim at coincidence of a font cache, and its effectiveness is large without additional hardware in respect of equipment cost.
[0095] Since the contents of each local font cache are mostly in agreement, even if assignment of what kind of process is performed, it is effective in the hit ratio of the cache beyond a certain level being securable.
[0096] Moreover, even if it is the case where there is no alphabetic data applicable to a local font cache, reference of the managed table of a global font cache is performed first, and if there is alphabetic data which other processes already created and was registered, it is usable for a **** reason. This has the effectiveness which prevents the duplication of processing whose multiple processes tend to read the outline font former data of the same alphabetic character in a hard disk drive unit, and has the effectiveness which cancels that an I/O process serves as a bottleneck as a result.

TECHNICAL FIELD

[Industrial Application] This invention relates to the alphabetic character data control equipment which performs parallel processing by two or more processors, and outputs printing data.

PRIOR ART

[Description of the Prior Art] In the airline printer used as peripheral devices, such as a workstation, the outline font which described the character outline independent of the resolution of an output unit as curvilinear information has been used from the font of the bit map form depending on the resolution of an output unit recently. However, in order to perform processing changed into the pixel data which finally suited resolution with each output unit, the conversion will take time amount. To the font which has a complicated profile like the kanji especially, it is still more so.
[0003] Then, about the alphabetic character [finishing / conversion] already processed by pixel data by the processor, the pixel data is temporarily stored in the memory called the font cache where it connects with the processor through the bus, and when the output request of the alphabetic character occurs again, the approach of using the pixel data is taken. Thereby, although the processing time in the case of printing can be shortened, it becomes important how only the alphabetic character with specific high operating frequency is held from there being a limitation in the capacity which it can have as a font cache. JP,64-88660,A and JP,2-233269,A are indicated as a management method of this font cache. Preventing un-arranging [which the former resets registration ranking when the alphabetic data registered into the font cache is called again, and is deleted as old data of a registration stage], the latter gives deletion priority

information beforehand to the font used.
[0004] However, even if it uses a font cache for max efficiently, with the equipment in a uniprocessor, a limitation will be too generated in improvement in the speed.
[0005] Then, the bus connection of two or more processors is carried out, and the way of thinking in which you are going to make it perform high-speed processing unrealizable by the uniprocessor is produced by the parallel processing by two or more processors. According to the computer system, there is what performs parallel processing by two or more processors, and each processor has already realized improvement in the speed of instruction access and a data access in activation of the assigned process by transmitting a part of run unit under current processing to cache memory.

EFFECT OF THE INVENTION

[Effect of the Invention] When the local font cache has been arranged [in / so that clearly / the processor of a multiprocessor configuration] to each microprocessor in the above example, it is easy to maintain the data consistency in a cache with the alphabetic character data control means of this invention. In addition, this invention is a solution means to aim at coincidence of a font cache, and its effectiveness is large without additional hardware in respect of equipment cost.
[0095] Since the contents of each local font cache are mostly in agreement, even if assignment of what kind of process is performed, it is effective in the hit ratio of the cache beyond a certain level being securable.
[0096] Moreover, even if it is the case where there is no alphabetic data applicable to a local font cache, reference of the managed table of a global font cache is performed first, and if there is alphabetic data which other processes already created and was registered, it is usable for a **** reason. This has the effectiveness which prevents the duplication of processing whose multiple processes tend to read the outline font former data of the same alphabetic character in a hard disk drive unit, and has the effectiveness which cancels that an I/O process serves as a bottleneck as a result.

TECHNICAL PROBLEM

[Problem(s) to be Solved by the Invention] To it, the processing changed into alphabetic data from an outline font as mentioned above is needed, and it does not go to the reason for saying that what is necessary is to program an activation process to **** like the former computer system, and just to transmit to cache memory by the airline printer, but the time amount spent on the writing to a font cache becomes large far with it. Therefore, it is necessary not to, perform transform processing from an outline font to alphabetic data if possible, but to fill the contents of the font cache with an airline printer by data with high operating frequency moreover.
[0007] Moreover, if each processor performs font cache management uniquely respectively and this alphabetic character is not registered into the font cache of another processor even if it is the alphabetic character already changed into alphabetic character pixel data by a certain processor, duplication processing which performs conversion to alphabetic character pixel data again is performed, and it is not efficient. When an activation process with many alphabetic character

kinds used is especially assigned till then to the processor which was low as for operating frequency suddenly, the utilization factor of a font cache will fall remarkably and will become a bottleneck on processing.
[0008] This invention is made in view of such a problem, and the place made into the purpose is to offer the alphabetic character data control equipment which raises nothing and the use effectiveness of a font cache for the contents of the font cache which each processor has as it is the same, and performs a high-speed document processing system.

MEANS

[Means for Solving the Problem] The share font cache where the alphabetic character data control equipment of this invention is accessible in common from two or more processors and these processors, An outline font feed zone accessible in common from said processor, It has an accessible partial font cache in said processor each. Reading from a local font cache is first tried to the acquisition demand of the alphabetic character at the time of a processes run. If it is, this will be read, and if there is nothing, when reading from a share font cache will be tried and there will be nothing further, former data are read from an outline font feed zone, alphabetic character pixel data are generated, and it registers with a share font cache.
[0010] The contents of the share font cache are transmitted to a local font cache, and it is characterized by updating the contents of each font cache at the process termination at the time of activation, or relaxation time.

OPERATION

[Function] According to this invention, the writing of new alphabetic character pixel data is performed to a share cache. Since data are transmitted to each local cache from a share cache at relaxation time, process termination **** can guarantee the identity of a local cache.

EXAMPLE

[Example] <u>Drawing 1</u> is the block block diagram of the airline printer of the electrophotography method held as one example of this invention.
[0013] An airline printer consists of three main parts. That is, they are the print engine 11, an engine control system 10, and a control circuit. The actuation of an airline printer is as follows. A printing control code is received from the interface 13 by which the control circuit was connected to devices, such as an external personal computer. The control processing program performed by the microprocessor unit 1 interprets this, generates a printing pixel, and writes in image memory 9. An engine control system 10 transmits this pixel data to an engine side synchronizing with actuation of the print engine 11, and performs printing processing.
[0014] This example takes in the means of this invention about the alphabetic character data control method of a control circuit among a series of above-mentioned processings.
[0015] The configuration of a control circuit is as follows. The microprocessor unit 1 of plurality

(drawing 1 four pieces) is connected to the global bus 3 by the bus mediation circuit 15. The bus mediation circuit 15 arbitrates contention of the microprocessor unit 1 to the global bus 3 according to the control signal of the bus mediation control circuit 12. Share ROM 5 and share RAM 4 are connected to the global bus 3. The control program of the airline printer of this example is written in the share ROM 5. Moreover, share RAM 4 is used for the working-level month memory of the above-mentioned control program, and the global font cache and receive data buffer which are mentioned later.

[0016] In the airline printer of this example, bit map printing, alphabetic printing, and easy graphic form printing are possible by assignment of a control code. However, in explanation of this example, since it is easy, only the case of alphabetic printing is explained.

[0017] The airline printer of this example performs alphabetic printing using an outline font. The former data of the outline font to be used are beforehand recorded on the hard disk drive unit 8. A hard disk drive unit 8 is controlled by I/O processor 6 through a disk controller 7. Moreover, the interface 13 of an external computer capital is also controlled by I/O processor 6. Since I/O processor 6 is accessed by the communication link port from the global bus 3, each microprocessor unit 1 is uniquely accessible to I/O processor 6.

[0018] The printing pixel which each microprocessor unit 1 generates is recorded on image memory 9. Image memory 9 is constituted by dual port memory, and it is timing different from mediation of the global bus 3, and the engine control circuit 10 reads pixel data, and it transmits it to the print engine 11.

[0019] On the other hand, the respectively local font cache 2 is connected to each microprocessor unit 1. Local bus connection of this font cache 2 is made, and as compared with the memory accessed through the global bus 3, since bus mediation is not needed, high-speed access can be performed.

[0020] Next, actuation of the font cache management method used by this example is explained using drawing 2 . In the font cache 2 locally connected to each microprocessor unit 1, the font managed table 206 and the font data section 201 are arranged. Moreover, in the global font cache field 315 placed after share RAM 4, the font managed table 206, the font data section 201, and an overflow area 205 are arranged. For the font managed table 206, in this example, 6 K bytes and the font data section 201 are [128 K bytes and an overflow area 205] 2 K bytes.

[0021] The DS on each [these] memory is the same also in the font cache 315 on share RAM 4 also in the local font cache 2. In drawing 2 , since it was easy, it considered as the explanatory view which accesses these DS with an address bus 215 and a data bus 216. In these actual bus access, local memory access is performed by the local bus of a microprocessor unit 1, and global access is performed by the global bus 3.

[0022] Such DS is explained separately below.

[0023] The font managed table 206 is an one-dimensional array which consists of 300 of the structure constituted by the alphabetic character assignment record 208 of 207 or 12 bytes of registration flag [6 bytes of], and 2 bytes of key 209. This structure was shown in drawing 6 . Here, an alphabetic character assignment record is DS which consists of the following elements.

[0024]

However, a typeface is a name used when a Hollerith type-like difference is specified, when two or more alphabetic character designs are performed about a certain alphabetic character. It is used in the above-mentioned DS, assigning a numeric value to this name. Moreover, character decoration assignment is not based on an alphabetic character typeface, but it is the assignment which makes the configuration of the target alphabetic character applicable [, such as an italic character and a void alphabetic character,] to actuation, and the internal representation in the case of using it by the above-mentioned DS is evaluated like the typeface.
[0025] The font data section 201 is memory with address arrangement [**** / finite length]. The interior is the set of the management unit memory 202 with a magnitude of 128 bytes. The management unit memory 202 consists of data division 204 with a magnitude of 126 bytes and an index part 203 with a magnitude of 2 bytes. The contents of the index part 203 are the indexes to the management unit memory which should follow a degree. Since the management unit memory 202 is magnitude immobilization, it can calculate the value of the real address by the easy operation from the value of an index. That is, when the value of an index is set to ix, the head address of the management unit memory 202 which should follow a degree is obtained as a value addr of a degree type.
[0026]
addr = ix x 128 + There is nothing to memory 201 head address ****, and a value 128 comes from the magnitude of management unit memory, and this operation is only an example of conversion to the real address. The index part 203 of the management unit memory 202 is one of the deformation of the so-called connection pointer.
[0027] the image data of an actual alphabetic character -- the data division 204 of the management unit memory 202 -- secondary -- it is used some and stored. The contents of the data stored are the number of the management unit memory 202 which 2 bytes of the beginning use, and the data following this are an image data. Moreover, all image datas consist of a data stream by which run length compression was carried out.
[0028] There is the approach of arranging a variable-length data stream dynamically on the continuous memory as the memory management technique as everyone knows. If this approach is used, after repeating the registration and deletion to a cache, the free area which is depended on the data stream which became unnecessary and which two or more magnitude does not have will arise. In order to rearrange such a fragmented memory resource that became unnecessary and to make the continuous big free area again, there is the need of performing processing which the execution time including the block move of memory requires. On the other hand, an intact part arises in the management unit memory of the last edge, and a means to use two or more fixed-length management unit memory which this example adopted is inferior to dynamic memory management in respect of a memory utilization factor. However, from management being simple,-izing can be carried out [hardware] easily, and also even if it performs a manager by software, it excels in the point which can perform high-speed processing.
[0029] An overflow area 205 is an one-dimensional array with a continuous magnitude of 2 K bytes.
[0030] The above font data sections 201, the font managed table 206, and an overflow area 205 are managed by a series of processing groups which each calls the cache quota processing 217. The read-out processing 210, the write-in processing 211, and the connection dissolution processing 212 are included in these processing group.
[0031] Next, the cache quota processing 217 is explained.

[0032] About the alphabetic character registered into the cache, the image data is divided and stored in two or more management unit memory 202 at the head in the management unit memory 202 of one of the font data sections 201. About the management unit memory 202 which comes first at this time, 300 management unit memory 202 is continuously used from the head of the font data section 201, respectively. Management unit memory 202 index of the beginning in the case of recording the data for a single character on two or more management unit memory 202 is specified by the key 209 in the structure of the font managed table 206. Although the font data section 201 has the magnitude of about 128 K bytes in the appearance described previously, this is 1024 sets of the management unit memory 202. Since there is a limit of such a size of an array, in this example, the image data per character is about an average of 430 bytes.
[0033] Next, the flow of processing of alphabetic character data acquisition is explained using drawing 5 .
[0034] When the demand which acquires alphabetic data occurs within a certain process, the alphabetic character data acquisition processing 501 is called. A demand is performed by alphabetic character assignment record which was mentioned above here. It is because it is necessary to treat the alphabetic character assignment from which magnitude, an alphabetic character kind, and the character decoration differ needless to say even if it is the same character code.
[0035] The procedure of the alphabetic character data acquisition processing 501 is as the flow chart of drawing 8 . The microprocessor unit 1 while performing processing searches the local font managed table 206 first (S801). If the same alphabetic character assignment record as the alphabetic character assignment record given here exists in the font managed table 206, alphabetic data (Hollerith type-like image data) exists in a local font cache. Therefore, read-out from a local font cache is performed (S802).
[0036] However, when the data of the alphabetic character which corresponds by retrieval of the local font managed table 206 are not able to be detected, the global font managed table 206 is searched continuously (S803). When the same alphabetic character assignment record as an alphabetic character assignment record exists in the font managed table 206, the value of the access counter 508 which is a global share variable is carried out +one (S804). An access counter 508 is a counter of 48 bit length, and is cleared at the time of a system startup. It is not cleared henceforth, but when there is an element newly registered into the font managed table 206 by the alphabetic character data acquisition processing 501 again when the candidate for retrieval was detected by the font managed table 206, +one is taken by the alphabetic data registration processing 502, respectively. Overflow is not generated in the busy condition of this example. The value of an access counter 508 is recorded on the registration flag 207 corresponding to the element with which the shared font managed table 206 was detected (S805). It can be judged that the value of an access counter 508 is the element accessed recently, so that this number is large, in order to count the count to which registration and read-out were performed.
[0037] Then, the key 209 of the element with which the font managed table 206 corresponds is taken out (S806), and alphabetic data read-out from a global font cache is performed (S807). On the global font managed table 206, when the element corresponding [an alphabetic character assignment record's] is not able to be detected, reading from the hard disk drive unit 8 of alphabetic data is required from a kernel (S808).
[0038] This is accompanied by the I/O process. In this example, between I/O processes, each microprocessor unit 1 does not wait for a processing result, but stops the process under current activation (S809). At this time, the update process 304 shown by drawing 3 is performed. Update

process 304 locks exclusively access of other microprocessor units 1 to a font cache field shared [RAM / 4] first. Next, all of the global font managed table 206 and the contents of the font data section 201 are transmitted to the local font managed table 206 and the font data section 201. In this example, this processing time is about 40m second. The exclusive control of access to the font cache field on share RAM 4 is canceled after this processing termination.
[0039] On the other hand, termination of reading of the outline font data from a hard disk drive unit 8 performs I/O termination processing by I/O processor 6. First, the flag which shows the kernel program actuation on a shared memory is checked, and there is a microprocessor unit while performing a kernel program, or detection is performed.
[0040] When the microprocessor unit 1 while performing a kernel program exists, I/O processor 6 carries out the I/O termination flag on a shared memory truly, and completes an I/O process. When the microprocessor unit 1 while executing a kernel program does not exist, I/O processor 6 sends an interruput signal to the waiting microprocessor unit 1 after carrying out the I/O termination flag on a shared memory truly using the interrupt control circuit 14. Here, the interrupt control circuit 14 of one waiting microprocessor unit waits to suspend processing and to stop activation of one of processes, when there is nothing.
[0041] When a kernel program is executed and termination of processing of I/O processor 6 is detected, process starting is urged to a kernel program by interruption to the waiting microprocessor unit 1, and the re-start process of a process stopped previously is performed (S810). However, the resumed process is not necessarily performed by the same microprocessor unit 1 as a pause or before.
[0042] The resumed process acquires the alphabetic data of an outline font by kernel service (S811), and generating processing of the image data showing the configuration of an alphabetic character is performed (S812). Although this processing is based on Hollerith type-like complexity, it is a kanji with a magnitude of 40x40 pixels, and this example takes the about [200m second] processing time from an average of 100m second. Then, the microprocessor unit 1 while performing a process registers the image of the generated alphabetic character only into a global font cache by the call of the alphabetic data registration processing 502 (S813).
[0043] In this example, the local font cache 2 does not write in except update process 304 as mentioned above. The local font cache 2 is always only set as the object of read-out.
[0044] Drawing 9 is the flow chart of processing of the alphabetic data registration processing 502.
[0045] From the alphabetic character data acquisition processing 501, the alphabetic data registration processing 502 is called as an argument, and the image data and the alphabetic character assignment record 208 of an alphabetic character are used. The registration processing 502 calls the table retrieval processing 213 (S901), searches the registration flag 207 of the font managed table 206, and takes out the smallest element of a value (S902). If the value of the registration flag 207 is 0 at this time, it will be judged that that element is an element which was not set as the object of font registration until now, or was just deleted. Then, the contents of the registration flag 207 are rewritten to FFFFFFFFFFFF of a hexadecimal. This is a temporary flag which shows that it is [current] under use. Then, the contents of the key 209 corresponding to this element are taken out, and write-in processing 211 is performed for the image data of a key and an alphabetic character to an argument (S903). As a return value of a processing result, when an overflow area error is returned, nothing is performed but registration processing is ended. In this case, at the time of registration processing termination, the value of the registration flag 207 is restored to the value before processing.

[0046] As a return value of the write-in processing 211, when a write error is returned, the registration processing 502 calls deletion 503 (S904). Since a free area arises in the font data section 201 of a font cache system as a result of this processing, the registration processing 502 calls the write-in processing 211 by making a key and an image data into an argument again. Furthermore, as a return value of a processing result, when a write error is returned, the same processing is repeated and registration is completed. In this case, +1 is carried out to the contents of the access counter 508 at the time of registration processing termination (S905), and this value is written in the registration flag 207 of the shared font managed table 206 (S906).
[0047] If this value is not 0 when the registration flag 207 of the font managed table 206 is searched and the smallest element of a value is taken out (S902), deletion 503 will be called (S907), a free area will be secured, and processing after S902 will be performed.
[0048] Drawing 10 is the flow chart of processing of the alphabetic character data deletion processing 503.
[0049] The alphabetic character data deletion processing 503 calls the table retrieval processing 213 (S1001), and takes out the minimum element which is not 0 about the registration flag 207 of the shared font managed table 206 (S1002), and the value of the key 209 corresponding to this element is acquired (S1003). Next, the connection dissolution processing 212 is called by making the value of this key into an argument (S1004), and processing is ended. As drawing 5 showed, the processing group of the cache quota processing 217 is called from the alphabetic character data acquisition processing 501, the alphabetic data registration processing 502, and the alphabetic character data deletion processing 503.
[0050] Next, the read-out processing 210, the write-in processing 211, and the connection dissolution processing 212 are explained among the cache quota processings 217, respectively.
[0051] Processing of the read-out processing 210 is explained using the flow chart of drawing 11
.
[0052] The read-out processing 210 takes out the key received as an argument (S1101), calls the address-arithmetic section 214 (S1102), and acquires the pointer to the management unit memory 202 of the font data section 201 (S1103). Then, according to this pointer, it accesses to the management unit memory 202 used as the head of image-data storing of an alphabetic character (S1104). According to the index to the management unit memory 202 connected below, the image data of an alphabetic character is read one after another (S1105). Since the already described contents of the data stored like are the number of the management unit memory 202 which 2 bytes of the beginning use, they can connect the management unit memory 202 of the required number. Moreover, the contents of the index part 203 of the management unit memory 202 located at the last of a series of connection are -1 (it is FFFF in a hexadecimal).
[0053] Processing of the write-in processing 211 is explained using the flow chart of drawing 12
.
[0054] The write-in processing 211 takes out the value of a key from the received argument (S1201), and calls the address translation processing 214 for this value to an argument (S1202). As this processing result, the pointer to the start address of the first usable management unit memory 202 is acquired in image-data writing (S1203). Next, the write-in processing 211 accesses this management unit memory 202 (S1204), and writes in the contents of the integral value (2 bytes) which broke the total data size of an image data by "the size -2 of management unit memory." This is in agreement with the total of the management unit memory 202 required for data storage. Next, an image data until it results in the magnitude of data division 204 is written in succeeding 2 bytes of this data (S1205). Further, if there is the remaining byte count of

an image data, write-in processing writes this data in an overflow area 205 first (S1206), it will be the font data section 201 interior of cache memory, and the intact management unit memory 202 will be looked for (S1207). When the intact management unit memory 202 is able to be discovered, the index to the current management unit memory 202 is written in the index part 203 of the management unit memory 202 accessed previously, and new association is generated (S1208). Then, from an overflow area 205, 126 bytes (this is the size of the data division 204 of the management unit memory 202) of data are taken out, and it writes in data division 204 (S1205). This is repeated until the contents of the overflow area 205 become empty or the intact management unit memory 202 is lost. When the remaining data of an overflow area 205 are lost, write-in processing is terminated normally. In this case, the contents of the index part 203 of the management unit memory 202 used at the end are rewritten to the value FFFF which shows joint termination, and the termination of association is generated (S1209).

[0055] When only the number which needs the intact management unit memory 202 cannot be detected on the other hand but the image data remains in the overflow area 205, the write-in processing 211 returns a write error to the processing by the side of a call (S1210), and stops processing. In this case, although it is necessary to remove the data of an unnecessary alphabetic character from the font data section 201 of font cache memory, to extend a free area, and to call the write-in processing 211 again, this processing is set and performed [judge and] at the call side (alphabetic data registration processing 502) which is high order processing.

[0056] Although an overflow area 205 is a memory area of the continuous address which has a sufficiently big capacity in the data of one character, when the image data of the alphabetic character of magnitude which has not gone into an overflow area 205 is passed, the write-in processing 211 returns an overflow area error (S1211), and stops processing. In this case, the image data of this alphabetic character is not registered into a font cache. Because, since especially the alphabetic character with a big image data has very low occurrence frequency, even if it registers with a cache, it is for not contributing to the processing speed improvement of a system.

[0057] Processing of the connection dissolution processing 212 is explained using the flow chart of drawing 13 .

[0058] The management unit memory 202 from which data were deleted, or the blank management unit memory 202 of data is distinguished from other management unit memory 202, when the value of an index part 203 is 0. The connection dissolution processing 212 acquires the key which is the value of the index from the argument to the management unit memory 202, and calls address translation processing 214 further (S1301). The address translation processing 214 changes a key value into the real address by the operation, and generates the pointer indicating the first management unit memory 202 on which the alphabetic data used as the candidate for deletion is recorded (S1302). Hereafter, the connection dissolution processing 212 calculates the real address of the head address of the following management unit memory 202 from the index part 203 of the management unit memory 202 which this pointer shows, and the contents of the following index part 203 are replaced by 0000 one after another (S1304). As long as connection of the management unit memory 202 continues, this actuation is repeated until it results in the management unit memory 202 whose contents of the index part 203 are FFFF (in hexadecimal). Then, the contents of the index part 203 of the last management unit memory 202 of connection are rewritten from FFFF to 0000 (S1305). (in hexadecimal)

[0059] Above, explanation of the access means to the data in the font cache which this example used is finished.

[0060] Some well-known approaches exist as a font cache memory management method which a font control mechanism uses. Fundamentally, font cache memory is memory assignment of finite length, and when it is newly going to register the image data of an alphabetic character and the free space of memory cannot be secured, it updates current registration. That is, from a cache field, unnecessary data (here image data of one of alphabetic characters) are deleted, and new data are registered. It is important as a management method to leave required data and to delete unnecessary data. However, in order that there may be no ideal approach of predicting whether the demand to what kind of alphabetic data will occur in the future, unnecessary data will be decided in accordance with a certain criteria. If this decision is appropriate, the probability (hit ratio) for the data of a request character to exist in a cache field will improve, and it will lead to improvement in processing speed. The typical management method which determines unnecessary data is following four.
[0061] Approach 1: It is judged that the data registered most in early stages are unnecessary in a current data constellation. The so-called approach of First In First Out (FIFO).
[0062] Approach 2: It is judged that the longest data in a current data constellation by which period use was not carried out are unnecessary. The so-called approach of Least Recently Used (LRU).
[0063] Approach 3: It is the approach judged that the data which were rather used for the degree and were used recently are [the longest data in a current data constellation by which period use was not carried out] unnecessary.
[0064] Approach 4: In a current data constellation, it is judged that data with the lowest operating frequency are unnecessary. The so-called approach of Least Frequently Used (LFU).
[0065] JP,64-88660,A" font cache control-system" uses the approach which has improved the approach 3 of facing an approach 1 among the conventional approaches. Moreover, JP,2-202464,A" airline printer" uses the approach 1.
[0066] These managements method is the approach invented to the single font control mechanism to the configuration which uses single font cache memory. By multimicroprocessor method of close coupling like this invention (based on a bus share), in order to establish data coincidence of each font cache, management of a local font cache and the shared font cache needs to be performed separately.
[0067] The approach of Least Recently Used (LRU) was used for this example in local management of a cache.
[0068] The parallel execution of each microprocessor unit 1 is controlled by the interruput signal which the interrupt control circuit 14 generates. This is explained using the explanatory view of drawing 3 , and the state transition diagram of drawing 4 below.
[0069] Parallel execution of the program performed in the airline printer control circuit of this example is carried out by two or more microprocessor units 1. At this time, the run unit of each processor is called a process. A process consists of a process header, a program object, and a working area. The value of a program counter in case a process is started (include the restart after interruption), the value of a stack pointer, the value of each register of a processor, memory size, memory management information, and the management information of a process proper are contained in a process header. The program code which can perform the object part of a process is contained. Moreover, the working area of a process is assigned to the variable within a process, a stack, etc. At the time of activation, the stereo of a process is arranged in a main storage area. In this example, the room 300 as shown in share RAM 4 at drawing 3 is taken, and processes 303 and 308 and 309 grades are arranged here. Since a process is generated by the

process generation demand to a kernel during program execution and it disappears by termination, the number of the process at the time of activation is unfixed.

[0070] Each microprocessor unit 1 which this example uses has a jump device to the specific address called some "exception-handling vectors." These are "the vector which should be jumped when interruption processing is performed", or "the vector which should be jumped after reset." All the microprocessor units 1 perform instruction word of the specific address specified by the vector after reset immediately after powering on. In the case of this example, this instruction word generates an endless loop and processing of each microprocessor unit will be in a standby condition. The field 301 arranged in the room 300 on the explanatory view of drawing 3 is a field where this endless loop is contained. This is called the standby routine 301 below.

[0071] Next, some interrupt control circuits 14 choose one microprocessor unit 1, and generate an interruput signal. Interruption is performed by specifying the jump place address at the time of interruption. The program counter of a microprocessor unit 1 is rewritten by this interruption by the value besides an endless loop. Here, the address which includes the jump instruction to the kernel field of an interruption dispatch table first is specified, and the jump to dispatch routine 302 occurs. Consequently, any one of four microprocessor units 1 starts the processing 306 of a kernel.

[0072] Actuation of the kernel processing 306 is shown in the flow chart of drawing 7 . Shortly after the processing is started by interruption, as for the kernel processing 306, the demand at the time of generating of interruption judges whether it is I/O termination processing (S701). If this decision is truth, I/O termination processing (S702) will be carried out. If it investigates whether a queue has a processing demand and there are S703 and a processing demand when decision of S701 is not truth, and when processing of S702 is completed, the contents of a demand will be processed (S704). Next, if the processing state of an I/O processor is acquired (S705) and the completed I/O process occurs, S701 or subsequent ones will be repeated. When all processing demands within a queue are processed, decision S703 becomes false, ends processing, and returns to the standby routine 301.

[0073] The microprocessor unit 1 which performs kernel processing assigns activation of the initiation process of a printing processing program once [of the beginning] after starting. The value of the starting address of a process is written in a dispatch table, and, specifically, interruption is generated in other microprocessor units 1 using the interrupt control circuit 14. The value of the bit to which this flag corresponds about the microprocessor unit 1 which has the flag 16 which shows the operating state of a microprocessor unit 1 in the interrupt control circuit 14, and is in the standby routine 301 is 1. Therefore, by the monitor of the value of this flag, the interrupt control circuit 14 specifies the waiting microprocessor unit 1, and sends an interruput signal to this. Since the jump place address is written in the dispatch table by other microprocessor units 1 working at a kernel, the program counter of the microprocessor unit 1 into which the interruput signal was inputted here is already changed into the appointed address by them.

[0074] Consequently, if starting of the first printing processing program is performed, all actuation will be performed by the back according to the flow of processing of this printing processing program. A printing processing program generates the process which receives a printing control code from an external computer through an interface 13, and performs reception printing processing for a printing control code. By this middle, Hollerith type-like generating, rearrangement of a printing pixel, a transfer of the data completion signal to an engine control system 10, etc. are performed.

[0075] The process whose control circuit of this example is a run unit respectively independent of each processing which was described here since it is a multimicroprocessor configuration is assigned. Especially, about alphabetic character generating, whenever there is an input of a line feed code, a process is generated. On the other hand, if a series of alphabetic character generating of all to a line feed code is performed, the process of this alphabetic character generating will serve as processing termination, and will carry out the ENQ of the process termination demand to the request queue of a kernel (the case where the case where an element is added to a queue is removed in an ENQ and an element is written to be a dequeue). Since the target process will be removed and discarded from room 300 if a kernel has this demand, the number of processes is [dozens of] at most. With the explanatory view of drawing 3 , the example of the room of a certain time amount and the condition of program execution is given. Three processes 303, 308, and 309, dispatch routine 302 and the demand generating processings 305 to a kernel, the kernel processing 306, I/O process 307, and update process 304 exist in room 300 in this case. The buffer 316 which, in addition to this, saves the field 315 of a global font cache (share) and the received data from an interface in room is arranged. Moreover, in drawing 3 , a rightward arrow head shows transition of the execution time, and a downward arrow head shows the direction of an upper address on memory arrangement. Furthermore, the polygonal lines 311, 312, and 313 show the value change of the program counter at the time of activation of a microprocessor unit 1, respectively.
[0076] In this example, update process 304 realizes the main actuation of this invention. An example is given below and this is explained. When the microprocessor unit 1 which has already processed the process 303 under activation tends to generate a new process, a process generation demand is given to a kernel. This demand is performed by calling the demand generating processing 305. It is shown that for this reason a program counter writes the polygonal line 312 to the location of the demand processing 305 temporarily, and it changes it. A process generation demand is performed in the location of an arrow head A. The ENQ of this demand is carried out to the request queue to a kernel. Kernel processing 306 is performed by another microprocessor unit 1. The kernel processing 306 (the polygonal line 313 shows) takes out this demand, newly generates a process 308, and writes that activation starting address in the dispatch table in dispatch routine 302.
[0077] The microprocessor unit 1 while performing kernel processing 306 uses the interrupt control circuit 14 next, specifies another microprocessor unit 1 under current standby, and sends an interruput signal. The polygonal line 311 shows the locus of the program counter of the microprocessor unit 1 which performs a process 308 as a result. This microprocessor unit 1 receives interruption, jumps it to dispatch routine 302 first, and is jumped to the starting address of a process 308.
[0078] When there is no waiting microprocessor unit 1 in the above-mentioned processing, it interrupts and generates and kernel processing assigns a processes run, when the microprocessor unit 1 which will be in a standby condition occurs.
[0079] The microprocessor unit 1 which is, on the other hand, performing processing previously shown with the polygonal line 312 generates an I/O process demand next in this example. As already explained in the flow chart of drawing 8 , the case where alphabetic data is read from a hard disk drive unit 8 etc. is accompanied by this processing. For this reason, the polygonal line 312 which is the locus of a program counter is again rewritten temporarily by the demand generating processing 305.
[0080] In this example, if all processes go into an I/O process, they will once stop continuation

of processing of a process and will return to a standby condition. A kernel can assign the microprocessor unit 1 included in a standby condition to activation of another process by this. It is because an I/O process produces the waiting about a number ms in many cases, so it is advantageous to other processes for use [move / activation] being [a processor resource] more efficient rather than a certain processor continues processing within a process in the meantime.
[0081] All processes perform update process 304, when interrupting a process as mentioned above, and when a process is completed. The contents of processing of update process 304 are recording the present condition of a process in process interruption, and updating the contents of the local font cache 2 of a microprocessor unit 1. As this was already described, from the font cache 315 on a shared memory, the contents of the font managed table 206 and the font data section 201 are read, and it realizes by transmitting to the local font cache 2.
[0082] Moreover, in process termination, the contents of processing of update process 304 open wide the memory which the process used, and the following is updating the contents of the local font cache 2 of a microprocessor unit 1 similarly.
[0083] The polygonal line 312 which is the locus of a program counter shows signs that the update process 304 is performed in the location of an arrow head B. This microprocessor unit 1 stops a process after that, and returns to activation of the standby routine 301 (see S809 of drawing 8). At this time, the value of the bit applicable to this microprocessor unit of the flag 16 of the interrupt control circuit 14 is set to 1, and it records going into a standby condition.
[0084] On the other hand, after an I/O process is completed with I/O processor 6, an interruput signal is sent to the waiting microprocessor unit 1 by the interrupt control circuit 14. Consequently, the microprocessor unit 1 to which the locus of a program counter is expressed with the polygonal line 312 starts activation of the kernel processing 306. As the flow chart of drawing 7 already explained, I/O termination processing is first performed in this case. Therefore, the polygonal line 312 goes into the field of I/O process 307 temporarily. The kernel processing which took out the processing result of I/O processor 6 here resumes the process 303 stopped like the point. Here, since the microprocessor unit 1 which showed the locus with the polygonal line 313 is in a standby condition, as for the kernel processing 306, the interrupt control circuit 14 performs interruption processing to this microprocessor unit 1. The specified microprocessor unit 1 passes along dispatch routine 302 by this interruption processing, and processing is resumed from the address which stopped last time [of a process 303] (see S810 of drawing 8). This is a location shown by the arrow head C.
[0085] The above is the flow of actuation of the parallel execution and the update process 304 in this example.
[0086] Next, another example is given about update process 304.
[0087] In the above-mentioned example, the update process 304 was what transmits all all the font managed tables 206 and font data sections 201 to the font managed table 206 and the font data section 201 of the local font cache 2 from the font cache 315 on a shared memory. In each processing of font reading and registration, in case this updates the registration flag 207 of the font managed table 206 after incrementing the access counter 508 which is a share variable, it is for operating it only to the global font managed table (share) 206.
[0088] However, the processing which transmits all the contents of the font data section 201 on a shared memory to partial memory is seen from the point of processing speed, and makes difficult use of the font cache more than a certain size. The more effective transfer should be performed to mount the font cache of very big size (1 M bytes or more). In order to realize this, it is possible to change the following part as another example.

[0089] First, it changes so that renewal of a registration flag (S805) may be performed to the both sides of the font managed font managed table 206 and 206 on the partial font cache 2 on a shared memory 4 in the flow chart of the alphabetic character data acquisition processing 501 of drawing 8 . Moreover, it changes so that renewal of a registration flag (S906) may be performed in the flow chart of the alphabetic data registration processing 502 of drawing 9 to the both sides of the font managed font managed table 206 and 206 on the partial font cache 2 on a shared memory 4.
[0090] This can perform update process 304, as shown in the flow chart of drawing 14 . That is, the contents of the font managed table 206 on a shared memory and the font managed table 206 on a font cache 2 are compared separately (S1401). When the registration flag 207 of the font managed table 206 on a shared memory is newer than the registration flag of the homotopic of the font managed table 206 on a font cache 2 (a value is large), it transmits this element (207,208,209 wholly) of the font managed table 206 of a shared memory to the homotopic of the font managed table 206 on a font cache 2 (S1402). Next, the key of this location is taken out and the address of the beginning of the management unit memory 202 of the starting position of the font data in the font data section 201 is calculated (S1403). Furthermore, an over-write [section / 201 / on a shared memory 4 / font data / them / the contents of the management unit memory 202 are read one by one according to a connection list about all the data of a single character, and / homotopic / of the font data section 201 on a font cache 2] (S1404). At this time, deletion is unnecessary. Then, it judges whether it is termination of the font managed table 206 (S1405), and subsequent (S1401) processings are repeated till table termination. In this processing, the processing time of update process 304 is shortened as the contents of the font cache are stabilized and the frequency of registration of a new alphabetic character falls.
[0091] By the process control accompanied by the update process 304 which was stated above, the contents of the local font cache 2 of two or more microprocessor units 1 can be made in agreement, respectively in the airline printer of this example. On the other hand, data reading from a font cache 2 can be performed at a high speed by not generating access to shared memory space during a processes run.
[0092] That is, that this example showed performs font registration accompanied by write-in actuation to a shared memory, and reading is a font management method performed only from partial memory. In case a font is registered into a font cache, in order not to perform the writing to partial memory, either, memory address management is easy and its constraint on arrangement of partial memory and a shared memory is flexible few.
[0093] Moreover, in the configuration in the above-mentioned example, it cannot be overemphasized by transposing the engine control system 10 of drawing 1 , and the part of the print engine 11 to a graphic controller, CRT equipment, etc., respectively that it is applicable to devices, such as a display and a workstation, as it is.

DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]
[Drawing 1] The block diagram of the example of this invention.
[Drawing 2] The explanatory view of a font cache system.
[Drawing 3] The explanatory view of multiprocessor processing.
[Drawing 4] The state transition diagram of multiprocessor processing.
[Drawing 5] The outline block diagram of alphabetic character data acquisition processing.
[Drawing 6] The DS explanatory view of an element in a font managed table.
[Drawing 7] The flow chart of kernel processing.
[Drawing 8] The flow chart of alphabetic character data acquisition processing.
[Drawing 9] The flow chart of alphabetic data registration processing.
[Drawing 10] The flow chart of alphabetic character data deletion processing.
[Drawing 11] The flow chart of read-out processing.
[Drawing 12] The flow chart of write-in processing.
[Drawing 13] The flow chart of connection dissolution processing.
[Drawing 14] The flow chart of another example of update process 304.
[Description of Notations]
1 -- Microprocessor unit
2 -- Font cache
3 -- Global bus
4 -- Share RAM
5 -- Share ROM
6 -- I/O processor
7 -- Disk controller
8 -- Hard disk drive unit
9 -- Image memory
10 -- Engine control system
11 -- Print engine
12 -- Bus mediation control circuit
13 -- Interface
14 -- Interrupt control circuit
15 -- Bus mediation circuit
16 -- Flag which shows the operating state of a microprocessor unit 1
201 -- Font data section
202 -- Management unit memory
203 -- Index part
204 -- Data division
205 -- Overflow area
206 -- Font managed table
207 -- Registration flag
208 -- Alphabetic character assignment record
209 -- Value of a key (index)
210 -- Read-out processing
211 -- Write-in processing

212 -- Deletion
300 -- Room
301 -- Standby routine
302 -- Dispatch routine
303, 308, 309 -- Process
304 -- Update process
305 -- Demand generating processing
306 -- Kernel processing
307 -- I/O process
315 -- Share font cache field
501 -- Alphabetic character data acquisition processing
502 -- Alphabetic data registration processing
503 -- Connection dissolution processing

乙第11号証

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報**（Ａ）　(11)特許出願公開番号

特開平5－261983

(43)公開日　平成５年(1993)10月12日

(51)Int.Cl.⁵		識別記号		庁内整理番号	F I			技術表示箇所
B 4 1 J	5/44			8907－2C				
	2/485							
G 0 6 F	3/12		G					
	15/16	3 2 0	K	8840－5L				
				8804－2C	B 4 1 J	3/ 12	A	

審査請求　未請求　請求項の数１(全 21 頁)　最終頁に続く

(21)出願番号　特願平4－62395

(22)出願日　平成４年(1992)３月18日

(71)出願人　000002369
セイコーエプソン株式会社
東京都新宿区西新宿２丁目４番１号

(72)発明者　長坂　文夫
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

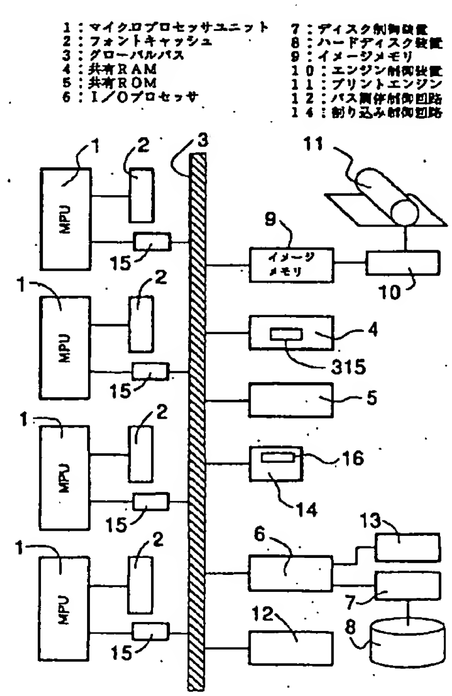
(74)代理人　弁理士　鈴木　喜三郎　(外１名)

(54)【発明の名称】　文字データ管理装置

(57)【要約】

【目的】　共有メモリ型密結合マルチマイクロプロセッサシステムで、各プロセッサが持つフォントキャッシュの内容を同一とし、各プロセッサ間に著しい処理速度の違いを生じさせない文字データ管理装置を提供すること。

【構成】　フォントキャッシュを、複数個のマイクロプロセッサ1についてそれぞれ局所的なメモリ2に配置し、共有ＲＡＭ4には共有フォントキャッシュ315を配置する。文字データの取得要求が発生した場合は、まず局所フォントキャッシュ2からの読み出しを試み、これが不可の時、次に共有フォントキャッシュ315からの読み込みを試み、さらにこれも不可であれば、アウトラインフォント元データからの文字発生をし、生成された文字形状のイメージデータは共有フォントキャッシュ315だけに書き込む。局所フォントキャッシュ2の更新は、プロセスのスイッチが行なわれる時点で行なう。



【特許請求の範囲】

【請求項１】　複数のプロセッサと、該プロセッサから共通にアクセス可能な共有フォントキャッシュと、前記プロセッサから共通にアクセス可能なアウトラインフォント供給部と、前記プロセッサ個々にアクセス可能な局所フォントキャッシュとを有し、

プロセス実行時の文字の取得要求に対し、まず局所的フォントキャッシュからの読み取りを試み、あればこれを読み取り、なければ共有フォントキャッシュからの読み取りを試み、更にない時は、アウトラインフォント供給部から元データを読み込み文字画素データを生成し、共有フォントキャッシュに登録する。実行時のプロセス終了あるいは休止時には、共有フォントキャッシュの内容を局所的フォントキャッシュに転送し、個々のフォントキャッシュの内容を更新することを特徴とする文字データ管理装置。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、複数のプロセッサにより並列処理を行い印字データを出力する文字データ管理装置に関する。

【０００２】

【従来の技術】ワークステーション等の周辺機器として用いられる印刷装置では、最近、出力装置の解像度に依存するビットマップ形のフォントから、出力装置の解像度に依存しない文字輪郭を曲線情報として記述したアウトラインフォントが使われてきている。しかし、最終的には各出力装置で解像度にあった画素データに変換する処理を行う為、その変換に時間がかかってしまう。特に、漢字の様な複雑な輪郭を持つフォントに対してはなおさらである。

【０００３】そこで、プロセッサにより既に画素データに処理された変換済みの文字に関しては、その画素データをバスを介してプロセッサに接続されているフォントキャッシュと呼ばれるメモリに一時的に蓄え、その文字の出力要求が再び発生した際にその画素データを使用する方法が採られている。それにより、印刷の際の処理時間が短縮可能であるが、フォントキャッシュとして持てる容量には限界があることから、特定の使用頻度の高い文字だけをいかに保持しておくかが重要となってくる。このフォントキャッシュの管理方法として、特開昭６４－８８６６０号及び特開平２－２３３２６９号が開示されている。前者はフォントキャシュに登録された文字データが再び呼び出された時、登録順位の再設定を行ない登録時期の古いデータとして削除される不都合を防ぐものであり、後者は予め使用フォントに削除優先情報を持たせるものである。

【０００４】しかし、フォントキャッシュを最大に効率良く利用しても、単一プロセッサでの装置では高速化にやはり限界が生じてしまう。

【０００５】そこで、複数のプロセッサをバス接続し、複数のプロセッサによる並列処理により、単一プロセッサで実現不可能な高速処理を行なわせようとする発想が生まれてくる。すでにコンピュータシステムでは、複数のプロセッサにより並列処理を行うものが有り、各々のプロセッサは割り当てられたプロセスの実行に当り、現在処理中の実行単位の一部をキャッシュメモリに転送することで命令アクセス、データアクセスの高速化を実現している。

【０００６】

【発明が解決しようとする課題】それに対し、印刷装置では、前述のようにアウトラインフォントから文字データに変換する処理を必要とし、前者のコンピュータシステムのような只単に実行プロセスのプログラムをキャッシュメモリに転送をすればよいと云うわけにはいかず、フォントキャッシュへの書き込みに費やされる時間は遥かに大きくなる。従って、印刷装置では、なるべくアウトラインフォントから文字データへの変換処理を行わず、しかもフォントキャッシュの内容を使用頻度の高いデータで満たしておくことが必要となる。

【０００７】また、個々のプロセッサが各々独自にフォントキャッシュ管理を行うと、あるプロセッサで既に文字画素データに変換された文字であっても別のプロセッサのフォントキャッシュに該文字が登録されていなければ、再度文字画素データへの変換を行う重複処理が行われ効率的でない。特に、それまで使用頻度の低かったプロセッサに、急に使用文字種の多い実行プロセスが割り当てられた場合においては、フォントキャッシュの利用率が著しく低下し処理上のボトルネックとなってしまう。

【０００８】本発明はこの様な問題に鑑みてなされたものであって、その目的とするところは、各プロセッサの持つフォントキャッシュの内容を同一となし、フォントキャッシュの利用効率を高め高速文書処理を行う文字データ管理装置を提供することにある。

【０００９】

【課題を解決するための手段】本発明の文字データ管理装置は、複数のプロセッサと、該プロセッサから共通にアクセス可能な共有フォントキャッシュと、前記プロセッサから共通にアクセス可能なアウトラインフォント供給部と、前記プロセッサ個々にアクセス可能な局所フォントキャッシュとを有し、プロセス実行時の文字の取得要求に対し、まず局所的フォントキャッシュからの読み取りを試み、あればこれを読み取り、なければ共有フォントキャッシュからの読み取りを試み、更にない時は、アウトラインフォント供給部から元データを読み込み文字画素データを生成し、共有フォントキャッシュに登録する。

【００１０】実行時のプロセス終了あるいは休止時には、共有フォントキャッシュの内容を局所的フォントキ

ャッシュに転送し、個々のフォントキャッシュの内容を更新することを特徴とする。

【0011】

【作用】本発明によれば、新たな文字画素データの書き込みは共有キャッシュに対し行われる。プロセス終了あるは休止時に共有キャッシュから個々の局部キャッシュへデータが転送されるから、局部キャッシュの同一性が保証できる。

【0012】

【実施例】図1はこの発明の一実施例として挙げた電子写真方式の印刷装置のブロック構成図である。

【0013】印刷装置は、3つの主要な部分から構成される。すなわち、プリントエンジン11、エンジン制御装置10および制御回路である。印刷装置の動作は、次の通りである。制御回路が外部のパーソナルコンピュータ等の機器に接続されたインターフェース13から、印刷制御コードを受信する。マイクロプロセッサユニット1により実行される制御処理プログラムがこれを解釈して印刷画素を生成し、イメージメモリ9に書き込む。エンジン制御装置10は、この画素データをプリントエンジン11の動作に同期してエンジン側に転送し印刷処理を行う。

【0014】この実施例は、上記の一連の処理の内、制御回路の文字データ管理方式に関して、本発明の手段を取り入れたものである。

【0015】制御回路の構成は次の通りである。複数（図1では4個）のマイクロプロセッサユニット1が、バス調停回路15によりグローバルバス3に接続される。バス調停回路15は、バス調停制御回路12の制御信号に従い、グローバルバス3に対するマイクロプロセッサユニット1の競合を調停する。グローバルバス3には共有ROM5、及び共有RAM4が接続される。共有ROM5には、本実施例の印刷装置の制御プログラムが書き込まれている。また共有RAM4は、上記制御プログラムの作業用メモリと、後述するグローバルフォントキャッシュおよび受信データバッファに使用される。

【0016】本実施例の印刷装置において、制御コードの指定によってビットマップ印刷、文字印刷、簡単な図形印刷が可能である。しかし本実施例の説明の中では簡単のため、文字印刷の場合についてのみ説明する。

【0017】本実施例の印刷装置は、アウトラインフォントを使用し文字印刷を行なう。使用するアウトラインフォントの元データは、ハードディスク装置8に予め記録されている。ハードディスク装置8は、ディスク制御装置7を介しI／Oプロセッサ6に制御される。また、外部コンピュータ都のインターフェース13もI／Oプロセッサ6により制御される。I／Oプロセッサ6はグローバルバス3から通信ポートによりアクセスされるため、個々のマイクロプロセッサユニット1は、独自にI／Oプロセッサ6にアクセス可能である。

【0018】各マイクロプロセッサユニット1が生成する印刷画素は、イメージメモリ9に記録される。イメージメモリ9はデュアルポートメモリによって構成され、エンジン制御回路10はグローバルバス3の調停とは別のタイミングで、画素データを読み出し、プリントエンジン11に転送する。

【0019】一方、各マイクロプロセッサユニット1には、それぞれ局所的なフォントキャッシュ2が接続される。このフォントキャッシュ2はローカルバス接続されており、グローバルバス3を介してアクセスされるメモリに比較し、バス調停を必要としないため高速なアクセスができる。

【0020】次に図2を用いて、本実施例で用いるフォントキャッシュ管理方法の動作を説明する。各マイクロプロセッサユニット1に局所的に接続されたフォントキャッシュ2には、フォント管理テーブル206及びフォントデータ部201が配置されている。また、共有RAM4の上に置かれるグローバルフォントキャッシュ領域315には、フォント管理テーブル206、フォントデータ部201及びあふれ領域205が配置される。本実施例においては、フォント管理テーブル206が6kバイト、フォントデータ部201が128kバイト、あふれ領域205が2kバイトである。

【0021】これら各メモリ上のデータ構造は、局所的なフォントキャッシュ2においても、共有RAM4上のフォントキャッシュ315においても同様である。図2では、簡単のためこれらデータ構造をアドレスバス215、データバス216によってアクセスする説明図とした。実際のこれらバスアクセスでは、局所的なメモリアクセスはマイクロプロセッサユニット1のローカルバスにより行なわれ、大域的なアクセスはグローバルバス3により行なわれる。

【0022】以下これらのデータ構造を個々に説明する。

【0023】フォント管理テーブル206は、6バイトの登録フラグ207、12バイトの文字指定レコード208及び2バイトのキー209によって構成される構造体の、300個からなる一次元配列である。この構造体を図6に示した。ここで、文字指定レコードは以下の要素からなるデータ構造である。

【0024】

データ構造の名称：	文字指定レコード	
要素1：	文字コード	2バイト
要素2：	書体指定	2バイト
要素3：	文字サイズ指定	2バイト
要素4：	文字修飾指定	6バイト

但し、書体は、ある文字について、複数の文字デザインが行われた場合、文字形状の差を特定する上で使用される呼称である。上記データ構造では、この呼称に数値を割り当てて使用する。また、文字修飾指定は、文字書体によらず、対象の文字の形状を、斜体文字、白抜き文字

などの操作対象とする指定であり、書体と同様に、上記データ構造で使用する場合の内部表現は数値化されている。

【００２５】フォントデータ部201は有限長の連続なアドレス配置を持つメモリである。内部は１２８バイトの大きさの管理単位メモリ202の集合である。管理単位メモリ202は、大きさ１２６バイトのデータ部204と、大きさ２バイトの指標部203からなる。指標部203の内容は、次に続くべき管理単位メモリへの指標である。管理単位メモリ202は大きさ固定であるため、指標の値から、実アドレスの値は簡単な演算により求めることができる。すなわち指標の値をixとしたとき、次に続くべき管理単位メモリ202の先頭番地は、次式の値addrとして得られる。

【００２６】

addr = ix × 128 ＋ メモリ201先頭番地

言うまでも無く、値128は管理単位メモリの大きさからくるもので、この演算は実アドレスへの変換の一例に過ぎない。管理単位メモリ202の指標部203は、いわゆる連結ポインタの変形の一つである。

【００２７】実際の文字のイメージデータは、管理単位メモリ202のデータ部204を副数個使用し格納される。格納されるデータの内容は、最初の２バイトが使用する管理単位メモリ202の個数であり、これに続くデータがイメージデータである。また全てのイメージデータはランレングス圧縮されたデータ列からなる。

【００２８】周知の様に、メモリ管理手法としては、可変長のデータ列を、連続したメモリ上に動的に配置する方法が有る。この方法を使用すると、キャッシュへの登録・削除を繰り返した後に、不要となったデータ列による、複数個の、大きさの一定でない空き領域が生じる。このような不要となった断片化したメモリ資源を再配置し、大きな連続した空き領域を再度作り出すためには、メモリのブロック移動を含む実行時間のかかる処理を行う必要が有る。これに対し、本実施例の採用した、固定長の管理単位メモリを複数利用する手段は、最終端の管理単位メモリに未使用部分が生じ、メモリ利用率の点で動的メモリ管理に劣るものである。しかし、管理が単純であることから、容易にハードウェア化できる他、ソフトウェアで管理プログラムを実行しても高速処理ができる点で優れている。

【００２９】あふれ領域205は、連続した２ｋバイトの大きさの一次元配列である。

【００３０】以上の様なフォントデータ部201、フォント管理テーブル206及びあふれ領域205は、いずれもキャッシュ割り当て処理217と呼ぶ一連の処理群によって管理される。これら処理群には読み出し処理210、書き込み処理211及び連結解消処理212が含まれる。

【００３１】次に、キャッシュ割り当て処理217について説明する。

【００３２】キャッシュに登録された文字については、そのイメージデータは、フォントデータ部201内のどれかの管理単位メモリ202を先頭に複数個の管理単位メモリ202に分割され格納される。このとき先頭となる管理単位メモリ202については、フォントデータ部201の先頭から連続して３００個の管理単位メモリ202をそれぞれ使用する。一文字分のデータを複数の管理単位メモリ202に記録する場合の最初の管理単位メモリ202指標は、フォント管理テーブル206の構造体の中のキー209によって指定する。先に述べた様にフォントデータ部201は約１２８ｋバイトの大きさを持つが、これは管理単位メモリ202の１０２４個の集合である。この様な配列の大きさの制限があるため本実施例では、１文字あたりのイメージデータは平均４３０バイト程度である。

【００３３】次に図５を用いて、文字データ取得の処理の流れを説明する。

【００３４】あるプロセス内で、文字データを取得する要求が発生した場合、文字データ取得処理501が呼び出される。ここで要求は、前述したような文字指定レコードによって行なわれる。言うまでもなく、同一文字コードであっても、大きさ、文字種、文字修飾が異なる文字指定を扱う必要があるためである。

【００３５】文字データ取得処理501の処理手順は、図８の流れ図の通りである。処理を実行中のマイクロプロセッサユニット1はまず局所的なフォント管理テーブル206を検索する(S801)。ここで与えられた文字指定レコードと同一の文字指定レコードがフォント管理テーブル206に存在すれば、文字データ（文字形状のイメージデータ）は局所的なフォントキャッシュに存在する。従って、局所的なフォントキャッシュからの読み出しが実行される(S802)。

【００３６】しかし、局所的なフォント管理テーブル206の検索で該当する文字のデータを検出できなかった場合は、続いて大域的なフォント管理テーブル206を検索する(S803)。文字指定レコードと同一の文字指定レコードがフォント管理テーブル206に存在した場合は、大域的な共有変数であるアクセスカウンタ508の値を＋１する(S804)。アクセスカウンタ508は４８ビット長のカウンタで、システム立ち上げ時にクリアされる。以降クリアされず、フォント管理テーブル206で検索対象が検出された場合文字データ取得処理501によって、またフォント管理テーブル206に新たに登録された要素が有った場合文字データ登録処理502によって、それぞれ＋１される。本実施例の使用状態では、オーバーフローは発生しない。アクセスカウンタ508の値は、共有のフォント管理テーブル206の、検出された要素に対応した登録フラグ207に記録される(S805)。アクセスカウンタ508の値は、登録、読み出しが行なわれた回数をカウントするため、この数が大きい程、最近アクセスされた要素であると判断できる。

【００３７】続いてフォント管理テーブル206の該当する要素のキー209が取り出され(S806),大域的なフォントキャッシュからの文字データ読み出しが行なわれる(S807)。大域的なフォント管理テーブル206で、文字指定レコードの一致する要素が検出できなかった場合は、カーネルに対し、文字データのハードディスク装置8からの読み込みを要求する(S808)。

【００３８】これは、Ｉ／Ｏ処理を伴う。本実施例では、Ｉ／Ｏ処理の間、各マイクロプロセッサユニット１は、処理結果を待たず現在実行中のプロセスを休止する(S809)。この時、図３で示す更新処理304が実行される。更新処理304は、まず共有ＲＡＭ4のフォントキャッシュ領域への他のマイクロプロセッサユニット1のアクセスを排他的にロックする。次に大域的なフォント管理テーブル206と、フォントデータ部201の内容を全て、局所的なフォント管理テーブル206とフォントデータ部201に転送する。本実施例においてこの処理時間は約４０ｍ秒である。この処理終了後、共有ＲＡＭ4上のフォントキャッシュ領域に対するアクセスの排他制御を解除する。

【００３９】一方、ハードディスク装置8からのアウトラインフォントデータの読み込みが終了すると、Ｉ／Ｏプロセッサ6によるＩ／Ｏ終了処理が行なわれる。まず、共有メモリ上のカーネルプログラム動作を示すフラグがチェックされ、カーネルプログラムを実行中のマイクロプロセッサユニットがあるか検出が行なわれる。

【００４０】カーネルプログラムを実行中のマイクロプロセッサユニット1が存在する場合、Ｉ／Ｏプロセッサ6は、共有メモリ上のＩ／Ｏ終了フラグを真にし、Ｉ／Ｏ処理を完了する。カーネルプログラムを実行中のマイクロプロセッサユニット1が存在しない場合は、Ｉ／Ｏプロセッサ6は、共有メモリ上のＩ／Ｏ終了フラグを真にしたの後、割り込み制御回路14を用いて、待機中のマイクロプロセッサユニット1に割り込み信号を送る。ここで、割り込み制御回路14は、待機中のマイクロプロセッサユニット1が無い場合、処理を保留しどれかのプロセスの実行が休止されるのを待つ。

【００４１】カーネルプログラムが実行され、Ｉ／Ｏプロセッサ6の処理の終了が検出された場合、カーネルプログラムは、待機中のマイクロプロセッサユニット1に対し、割り込みによってプロセス起動を促し、先に休止したプロセスの再開処理を行なう(S810)。但し、再開されたプロセスが、休止以前と同じマイクロプロセッサユニット1により実行されるとは限らない。

【００４２】再開されたプロセスは、カーネルサービスによりアウトラインフォントの文字データを取得し(S811)、文字の形状を現すイメージデータの発生処理を行なう(S812)。この処理は文字形状の複雑さによるが、本実施例では４０×４０画素の大きさの漢字で、平均１００ｍ秒から２００ｍ秒程度の処理時間を要する。この後、プロセスを実行中のマイクロプロセッサユニット1は、生成された文字のイメージを、文字データ登録処理502の呼び出しにより、大域的なフォントキャッシュにだけ登録する(S813)。

【００４３】以上の様に本実施例においては、局所的なフォントキャッシュ2は、更新処理304以外で書き込みを行なわない。局所的なフォントキャッシュ2は、常に読み出しの対象となるだけである。

【００４４】図９は、文字データ登録処理502の処理の流れ図である。

【００４５】文字データ登録処理502は、文字データ取得処理501から文字のイメージデータと、文字指定レコード208を引数として呼び出され使用される。登録処理502は、テーブル検索処理213を呼び出し(S901)、フォント管理テーブル206の登録フラグ207を検索し、値の最も小さい要素を取り出す(S902)。このとき、登録フラグ207の値が０であれば、その要素は、今までフォント登録の対象とならなかったか、あるいは削除されたばかりの要素であると判断する。そこで、登録フラグ207の内容を１６進数のFFFFFFFFFFFFに書き換える。これは現在使用中であることを示す一時的フラグである。続いて、この要素に対応するキー209の内容を取り出し、キーと文字のイメージデータを引数に書き込み処理211を実行する(S903)。処理結果の戻り値として、あふれ領域エラーが戻された場合は、何も行なわず、登録処理を終了する。この場合、登録処理終了時には、登録フラグ207の値を、処理前の値に復元する。

【００４６】書き込み処理211の戻り値として、書き込みエラーが戻された場合は、登録処理502は削除処理503を呼び出す(S904)。この処理の結果、フォントキャッシュシステムのフォントデータ部201には空き領域が生じるため、登録処理502は再び、キーとイメージデータを引数として、書き込み処理211を呼び出す。更に、処理結果の戻り値として、書き込みエラーが戻された場合は、同様の処理を繰り返し、登録が完了する。この場合は、登録処理終了時にアクセスカウンタ508の内容に＋１を行ない(S905)、この値を共有のフォント管理テーブル206の登録フラグ207に書き込む(S906)。

【００４７】フォント管理テーブル206の登録フラグ207を検索し、値の最も小さい要素を取り出したとき(S902)、この値が０でなければ、削除処理503を呼び出し(S907)、空き領域を確保し、S902以降の処理を行なう。

【００４８】図１０は、文字データ削除処理503の処理の流れ図である。

【００４９】文字データ削除処理503は、テーブル検索処理213を呼び出し(S1001)、共有のフォント管理テーブル206の登録フラグ207について、０でない最小の要素を取り出し(S1002)、この要素に対応したキー209の値を得る(S1003)。次にこのキーの値を引数として、連結解消処理212を呼び出し(S1004)処理を終了する。図５で示し

たように、文字データ取得処理501、文字データ登録処理502および文字データ削除処理503から呼び出されるのは、キャッシュ割り当て処理217の処理群である。

【００５０】次に、キャッシュ割り当て処理217の内、読み出し処理210、書き込み処理211及び連結解消処理212についてそれぞれ説明する。

【００５１】読み出し処理210の処理を図１１の流れ図を用いて説明する。

【００５２】読み出し処理210は引数として受け取ったキーを取り出し(S1101)アドレス演算部214を呼び出し(S1102)、フォントデータ部201の管理単位メモリ202へのポインタを取得する(S1103)。続いてこのポインタに従い、文字のイメージデータ格納の先頭となる管理単位メモリ202へアクセスする(S1104)。以下連結した管理単位メモリ202への指標に従い文字のイメージデータを次々に読み出す(S1105)。既に述べた様に、格納されるデータの内容は、最初の２バイトが使用する管理単位メモリ202の個数であるため、必要な個数の管理単位メモリ202を連結することができる。また、一連の連結の最終に位置する管理単位メモリ202の指標部203の内容は－１（１６進数でFFFF）である。

【００５３】書き込み処理211の処理を図１２の流れ図を用いて説明する。

【００５４】書き込み処理211は、受け取った引数からキーの値を取り出し(S1201)、この値を引数にアドレス変換処理214を呼び出す(S1202)。この処理結果として、イメージデータ書き込みに使用可能な最初の管理単位メモリ202の先頭アドレスへのポインタを取得する(S1203)。次に、書き込み処理211は、この管理単位メモリ202にアクセスし(S1204)、イメージデータの全データサイズを”管理単位メモリのサイズ－２”で割った整数値（２バイト）の内容を書き込む。これはデータ格納に必要な管理単位メモリ202の総数に一致する。次に、この２バイトのデータに連続して、データ部204の大きさに至るまでのイメージデータを書き込む(S1205)。もし更に、イメージデータの残りバイト数があれば、書き込み処理はこのデータをまずあふれ領域205に書き込み(S1206)、キャッシュメモリのフォントデータ部201内部で、未使用の管理単位メモリ202を捜す(S1207)。未使用の管理単位メモリ202が発見できた場合は、先にアクセスした管理単位メモリ202の指標部203に、現在の管理単位メモリ202への指標を書き込み、新たな結合を生成する(S1208)。続いてあふれ領域205から、１２６バイト（これは、管理単位メモリ202のデータ部204のサイズである）のデータを取り出し、データ部204に書き込む(S1205)。これを、あふれ領域205の内容が空になるか、未使用の管理単位メモリ202が無くなるまで繰り返す。あふれ領域205の残りデータが無くなった場合、書き込み処理は正常終了する。この場合、最後に使用した管理単位メモリ202の指標部203の内容を、結合終了を示す値FFFFに書き換え、結合の終端を生成する(S1209)。

【００５５】一方、未使用の管理単位メモリ202が必要な数だけ検出できず、あふれ領域205にイメージデータが残っている場合は、書き込み処理211は、呼び出し側の処理に書き込みエラーを返し(S1210)処理を中止する。この場合、不要な文字のデータをフォントキャッシュメモリのフォントデータ部201から取り除き、空き領域を広げ、再度書き込み処理211を呼び出す必要があるが、この処理は、上位処理である呼び出し側（文字データ登録処理502）において判断、実行される。

【００５６】あふれ領域205は、１文字のデータには充分大きな容量を持つ連続したアドレスのメモリ領域であるが、もしあふれ領域205に入りきらない大きさの文字のイメージデータが渡された場合は、書き込み処理211はあふれ領域エラーを返し(S1211)処理を中止する。この場合、この文字のイメージデータはフォントキャッシュに登録されない。なぜなら、特に大きなイメージデータを持つ文字は、発生頻度が極めて低いため、キャッシュに登録しても、システムの処理速度改善に寄与しないためである。

【００５７】連結解消処理212の処理を図１３の流れ図を用いて説明する。

【００５８】データの削除された管理単位メモリ202、あるいはデータの空白な管理単位メモリ202は、指標部203の値が０であることによって他の管理単位メモリ202から区別される。連結解消処理212は、引数から管理単位メモリ202への指標の値であるキーを取得し、さらにアドレス変換処理214の呼び出しを行なう(S1301)。アドレス変換処理214は、キー値を演算により実アドレスに変換し、削除対象となる文字データの記録されている最初の管理単位メモリ202を指し示すポインタを発生する(S1302)。以下、連結解消処理212は、このポインタが示す管理単位メモリ202の指標部203から次の管理単位メモリ202の先頭番地の実アドレスを計算し、後に続く指標部203の内容を次々に0000で置き換える(S1304)。この操作を管理単位メモリ202の連結が続く限り、指標部203の内容が（１６進数で）FFFFである管理単位メモリ202に至るまで繰り返す。このあと、連結の最終の管理単位メモリ202の指標部203の内容を、（１６進数で）FFFFから0000に書き換える(S1305)。

【００５９】以上で、本実施例が使用したフォントキャッシュ内のデータへのアクセス手段の説明を終わる。

【００６０】フォント管理機構が使用するフォントキャッシュメモリ管理方法として、いくつか公知の方法が存在する。基本的にフォントキャッシュメモリは、有限長のメモリ割り当てであり、新たに文字のイメージデータを登録しようとし、メモリの空きスペースが確保できない場合は、現在の登録を更新する。すなわち、キャッシュ領域から、不要なデータ（ここではどれかの文字のイメージデータ）を削除し、新規のデータを登録する。管

理方法として重要なのは、必要なデータを残し、不要なデータを削除することである。しかし、将来どのような文字データに対する要求が発生するかを予測する理想的な方法が無いため、何らかの基準に従い不要なデータを決めることになる。この決定が適切であれば、要求文字のデータが、キャッシュ領域に存在する確率（ヒット率）が向上し、処理速度の向上につながる。不要なデータを決める代表的な管理方法は以下の４つである。

【００６１】方法１：現在のデータ群の中で、最も初期に登録されたデータが不要と判断される。いわゆるFirst In First Out(FIFO)の方法。

【００６２】方法２：現在のデータ群の中で、最も長い期間使用されなかったデータが不要と判断される。いわゆるLeast Recently Used(LRU)の方法。

【００６３】方法３：現在のデータ群の中で、最も長い期間使用されなかったデータはむしろ次に使用され、最近使用されたデータが不要と判断される方法。

【００６４】方法４：現在のデータ群の中で、最も使用頻度の低いデータが不要と判断される。いわゆるLeast Frequently Used(LFU)の方法。

【００６５】従来方法の内、例えば特開昭６４－８８６６０”フォントキャッシュ制御方式”は方法１に対し方法３の改善を行った方法を用いている。また特開平２－２０２４６４”印刷装置”は方法１を用いている。

【００６６】これら管理方式は、単一のフォント管理機構に対し、単一のフォントキャッシュメモリを使用する構成に対し発明された方法である。本発明の様な（バス共有による）密結合のマルチマイクロプロセッサ方式では、各フォントキャッシュのデータ一致を確立するため、ローカルなフォントキャッシュと、共有されたフォントキャッシュの管理が個々に行なわれる必要がある。

【００６７】本実施例は、キャッシュの局所的な管理においてはLeast Recently Used(LRU)の方法を用いた。

【００６８】各マイクロプロセッサユニット1の並列実行は、割り込み制御回路14の発生する割り込み信号によって制御される。以下図３の説明図と図４の状態遷移図を用いて、これを説明する。

【００６９】本実施例の印刷装置制御回路において実行されるプログラムは、複数のマイクロプロセッサユニット1により並列実行される。この時、個々のプロセッサの実行単位をプロセスと呼ぶ。プロセスは、プロセスヘッダ、プログラムオブジェクト、作業領域から構成される。プロセスヘッダには、プロセスが（中断後の再開を含め）開始される場合のプログラムカウンタの値、スタックポインタの値、プロセッサの各レジスタの値、メモリサイズ、メモリ管理情報及びプロセス固有の管理情報が含まれる。プロセスのオブジェクト部分は実行可能なプログラムコードが含まれる。また、プロセスの作業領域は、プロセス内の変数、スタック等に割り当てられる。実行時にはプロセスの実体は主記憶領域に配置される。本実施例では、共有ＲＡＭ4に図３に示す様なメモリ空間300をとり、ここにプロセス303、308、309等を配置する。プロセスはプログラム実行中にカーネルに対するプロセス生成要求で生成され、終了により消失するので、実行時のプロセスの個数は不定である。

【００７０】本実施例の使用する各マイクロプロセッサユニット1は、幾つかの”例外処理ベクタ”と呼ばれる特定アドレスへのジャンプ機構を持っている。これらは、例えば”割り込み処理が行なわれた場合ジャンプすべきベクタ”、あるいは”リセット後にジャンプすべきベクタ”などである。電源投入直後全てのマイクロプロセッサユニット1は、リセット後のベクタで指定された特定のアドレスの命令語を実行する。本実施例の場合、この命令語は無限ループを生成し、各マイクロプロセッサユニットの処理は待機状態となる。図３の説明図上のメモリ空間300に配置された領域301は、この無限ループが含まれる領域である。以下これを待機ルーチン301と呼ぶ。

【００７１】次に、割り込み制御回路１４が、どれか一つのマイクロプロセッサユニット1を選び、割り込み信号を発生する。割り込みは、割り込み時のジャンプ先アドレスを指定し行なわれる。この割り込みによってマイクロプロセッサユニット1のプログラムカウンタは、無限ループの外の値に書き換えられる。ここではまず割り込みディスパッチテーブルのカーネル領域へのジャンプ命令を含むアドレスが指定され、ディスパッチルーチン302へのジャンプが発生する。この結果、４個有るマイクロプロセッサユニット1の内のどれか一つは、カーネルの処理306を開始する。

【００７２】カーネル処理306の動作は、図７の流れ図に示される。カーネル処理306は、割り込みによってその処理が開始されると直ちに、割り込みの発生時の要求がＩ／Ｏ終了処理であるか判断する(S701)。この判断が真であれば、Ｉ／Ｏ終了処理(S702)が実施される。S701の判断が真でない場合およびS702の処理が終了した場合は、待ち行列に処理要求があるか調べS703、処理要求が有れば、要求内容を処理する(S704)。次にＩ／Ｏプロセッサの処理状態を取得し(S705)、完了したＩ／Ｏ処理があれば、S701以降を繰り返す。待ち行列内の全ての処理要求を処理したときは、判断S703が偽となり、処理を終了し待機ルーチン301に戻る。

【００７３】カーネル処理を実行するマイクロプロセッサユニット1は、起動後最初の１回は、印刷処理プログラムの開始プロセスの実行を割り当てる。具体的には、プロセスの開始アドレスの値をディスパッチテーブルに書き込み、割り込み制御回路14を使用し他のマイクロプロセッサユニット1に割り込みを発生する。割り込み制御回路14には、マイクロプロセッサユニット1の動作状態を示すフラグ16が有り、待機ルーチン301にあるマイクロプロセッサユニット1についてはこのフラグの対応

するビットの値が1である。従って割り込み制御回路14はこのフラグの値の監視により、待機中のマイクロプロセッサユニット1を特定し、これに割り込み信号を送る。既にカーネルで動作中の他のマイクロプロセッサユニット1によって、ディスパッチテーブルにジャンプ先アドレスが書き込まれているため、ここで割り込み信号が入力されたマイクロプロセッサユニット1のプログラムカウンタは、指定アドレスに変更される。

【0074】この結果、最初の印刷処理プログラムの起動が行なわれると、後はこの印刷処理プログラムの処理の流れに従い、全ての動作が行なわれる。印刷処理プログラムは、インターフェース13を介し外部コンピュータから印刷制御コードを受信するプロセスを生成し、印刷制御コードを受け取り印刷処理を行なう。この途中で、文字形状発生、印刷画素の並べ換え、エンジン制御装置10へのデータ完了信号の転送などが行なわれる。

【0075】本実施例の制御回路が、マルチマイクロプロセッサ構成であることから、ここに述べた様な個々の処理にはそれぞれ独立の実行単位であるプロセスが割り当てられる。特に、文字発生に関しては、改行コードの入力が有る毎にプロセスの生成を行なう。一方で、この文字発生のプロセスは、改行コードまでの、一連の文字発生を全て行なうと、処理終了となり、プロセス終了要求をカーネルの要求待ち行列にエンキュウする（待ち行列に要素を追加する場合をエンキュウ、要素が除かれる場合をデキュウと書く）。カーネルはこの要求があると対象のプロセスをメモリ空間300から取り除き廃棄するので、プロセスの数は高々数十個である。 図3の説明図では、ある時間のメモリ空間とプログラム実行の状態の例を挙げている。メモリ空間300には、この場合3つのプロセス303、308、309と、ディスパッチルーチン302、カーネルへの要求発生処理305、カーネル処理306、I／O処理307及び更新処理304が存在する。メモリ空間にはこの他に、大域的な（共有の）フォントキャッシュの領域315、インターフェースからの受信データを保存するバッファ316が配置される。また、図3において右向きの矢印は実行時間の推移を示し、下向きの矢印はメモリ配置上の上位アドレス方向を示す。さらに折れ線311、312、313はそれぞれマイクロプロセッサユニット1の実行時のプログラムカウンタの値の変化を示す。

【0076】この実施例において本発明の主な動作を実現するのは更新処理304である。以下例を挙げこれを説明する。既に実行中のプロセス303を処理しているマイクロプロセッサユニット1が、新たなプロセスを生成しようとする場合、プロセス生成要求をカーネルに対し行なう。この要求は要求発生処理305を呼び出して行なわれる。折れ線312は、このためプログラムカウンタが一時的に要求処理305の位置に書き変わることを示している。矢印Aの位置でプロセス生成要求を行なう。この要求は、カーネルに対する要求待ち行列にエンキュウされる。カーネル処理306は別のマイクロプロセッサユニット1により実行されている。カーネル処理306（折れ線313で示す）はこの要求を取り出し、新たにプロセス308を生成し、その実行開始番地をディスパッチルーチン302内にあるディスパッチテーブルに書き込む。

【0077】カーネル処理306を実行中のマイクロプロセッサユニット1は次に割り込み制御回路14を用い、現在待機中の別のマイクロプロセッサユニット1を特定し、割り込み信号を送る。折れ線311はこの結果プロセス308を実行するマイクロプロセッサユニット1のプログラムカウンタの軌跡を示している。このマイクロプロセッサユニット1は、割り込みを受けまずディスパッチルーチン302へジャンプし、プロセス308の実行開始アドレスにジャンプする。

【0078】上記処理において待機中のマイクロプロセッサユニット1が無い場合、カーネル処理は、待機状態になるマイクロプロセッサユニット1が発生した時点で、割り込み発生しプロセス実行を割り当てる。

【0079】一方、先に折れ線312で示される処理を実行しているマイクロプロセッサユニット1は、この例では次にI／O処理要求を発生する。既に図8の流れ図において説明したように、文字データをハードディスク装置8から読み込む場合等が、この処理を伴う。このためプログラムカウンタの軌跡である折れ線312は、再び要求発生処理305に一時的に書き換えられる。

【0080】本実施例において、全てのプロセスはI／O処理に入るとプロセスの処理の継続を一旦中止し、待機状態に戻る。これによってカーネルは、待機状態に入ったマイクロプロセッサユニット1を、別のプロセスの実行に割り当てることができる。なぜなら、I／O処理は多くの場合数ミリ秒程度の待ちを生じるため、この間、あるプロセッサがプロセス内で処理を継続するより、他のプロセスに実行を移す方がプロセッサ資源の効率的な使用にとり有利だからである。

【0081】全てのプロセスは、上記の様にプロセスを中断する場合、及びプロセスが終了した場合に、更新処理304を実行する。更新処理304の処理内容は、プロセス中断の場合は、プロセスの現在の状態を記録し、マイクロプロセッサユニット1の局所的なフォントキャッシュ2の内容を更新することである。これは既に述べたように、共有メモリ上のフォントキャッシュ315から、フォント管理テーブル206とフォントデータ部201の内容を読み出し、局所的なフォントキャッシュ2へ転送することで実現される。

【0082】また、プロセス終了の場合は、更新処理304の処理内容は、プロセスの使用したメモリを開放し、以下は同様にマイクロプロセッサユニット1の局所的なフォントキャッシュ2の内容を更新することである。

【0083】プログラムカウンタの軌跡である折れ線312は、矢印Bの位置で更新処理304を実行している様子を

示す。このマイクロプロセッサユニット1は、その後プロセスを休止し待機ルーチン301の実行に戻る（図８のS809を見よ）。この時、割り込み制御回路14のフラグ16のこのマイクロプロセッサユニットに該当するビットの値を１とし、待機状態に入ることを記録する。

【００８４】一方、Ｉ／Ｏプロセッサ6によりＩ／Ｏ処理が終了すると、割り込み制御回路14によって待機中のマイクロプロセッサユニット1に割り込み信号が送られる。この結果、折れ線312でプログラムカウンタの軌跡が現されるマイクロプロセッサユニット1がカーネル処理306の実行に入る。図７の流れ図で既に説明したように、この場合まずＩ／Ｏ終了処理が行なわれる。従って折れ線312は一時的にＩ／Ｏ処理307の領域に入る。ここでＩ／Ｏプロセッサ6の処理結果を取り出したカーネル処理は、先ほど休止したプロセス303を再開する。ここでは折れ線313で軌跡を示したマイクロプロセッサユニット1が待機状態にあるため、カーネル処理306は割り込み制御回路14によってこのマイクロプロセッサユニット１に割り込み処理を行なう。この割り込み処理によって、指定されたマイクロプロセッサユニット1はディスパッチルーチン302を通り、プロセス303の前回休止した番地から処理を再開する（図８のS810を見よ）。これは矢印Ｃで示される位置である。

【００８５】以上が本実施例における並列実行及び更新処理304の動作の流れである。

【００８６】次に、更新処理304について、別の実施例をあげる。

【００８７】前述の実施例において、更新処理304は、共有メモリ上のフォントキャッシュ315から、フォント管理テーブル206の全てと、フォントデータ部201の全てを局所的なフォントキャッシュ2のフォント管理テーブル206およびフォントデータ部201に転送するものであった。これは、フォント読み込み、登録の各処理において、共有変数であるアクセスカウンタ508をインクリメントした後、フォント管理テーブル206の登録フラグ207を更新する際に、大域的な（共有の）フォント管理テーブル206に対してだけ操作を行なうためである。

【００８８】しかし、共有メモリ上のフォントデータ部201の内容全部を、局所メモリに転送する処理は、処理速度の点から見て、あるサイズ以上のフォントキャッシュの使用を困難とする。極めて大きなサイズ（１Ｍバイト以上）のフォントキャッシュを実装したい場合、より効果的な転送を行なうべきである。これを実現するために、別の実施例として次の部分を変更することが考えられる。

【００８９】まず図８の文字データ取得処理501の流れ図の中で、登録フラグ更新(S805)を、共有メモリ4上のフォント管理テーブル206及び局所フォントキャッシュ2上のフォント管理テーブル206の双方に対して行なう様に変更する。また、図９の文字データ登録処理502の流れ図の中で、登録フラグ更新(S906)を、共有メモリ4上のフォント管理テーブル206及び局所フォントキャッシュ2上のフォント管理テーブル206の双方に対して行なう様に変更する。

【００９０】これによって更新処理304は図１４の流れ図の様に行なうことができる。すなわち、共有メモリ上のフォント管理テーブル206と、フォントキャッシュ2上のフォント管理テーブル206の内容を個々に比較する(S1401)。共有メモリ上のフォント管理テーブル206の登録フラグ207が、フォントキャッシュ2上のフォント管理テーブル206の同位置の登録フラグより新しい（値が大きい）場合は、共有メモリのフォント管理テーブル206のこの要素（207,208,209全て）を、フォントキャッシュ2上のフォント管理テーブル206の同位置に転送する(S1402)。次にこの位置のキーを取り出し、フォントデータ部201でのフォントデータの開始位置の管理単位メモリ202の最初のアドレスを計算する(S1403)。さらに、共有メモリ4上のフォントデータ部201から、管理単位メモリ202の内容を、一文字のデータ全てについて連結リストに従い順次読み出し、フォントキャッシュ2上のフォントデータ部201の同位置にオーバーライトする(S1404)。この時、削除処理は不要である。続いてフォント管理テーブル206の終了かどうかを判断し(S1405)、（Ｓ１４０１）以降の処理をテーブル終了まで繰り返す。この処理では、フォントキャッシュの内容が安定し新たな文字の登録の頻度が下がるにつれ、更新処理３０４の処理時間が短縮される。

【００９１】以上に述べた様な更新処理304を伴うプロセス管理によって、本実施例の印刷装置においては、複数個のマイクロプロセッサユニット1の局所的なフォントキャッシュ2の内容を、それぞれ一致させることができる。この一方で、プロセス実行中はフォントキャッシュ2からのデータ読み込みは、共有メモリ空間へのアクセスを発生しないことによって高速に行なうことができる。

【００９２】すなわち本実施例が示したのは、書き込み動作を伴うフォント登録は共有メモリに行ない、読み込みは局所メモリだけから行なうフォント管理方法である。フォントをフォントキャッシュに登録する際、局所メモリへの書き込みも行なわないため、メモリアドレス管理は簡単であり、局所メモリ、共有メモリの配置上の制約が少なく柔軟である。

【００９３】また、上記実施例での構成において、図１のエンジン制御装置10及びプリントエンジン11の部分を、グラフィックコントローラおよびＣＲＴ装置等にそれぞれ置き換えることによって、そのまま表示装置、ワークステーション等の機器に適用できることは言うまでもない。

【００９４】

【発明の効果】以上の実施例において明らかなように、

マルチプロセッサ構成の処理装置において、個々のマイクロプロセッサに局所的なフォントキャッシュを配置した場合、本発明の文字データ管理手段によって、キャッシュ内のデータ一致性を保つのは容易である。加えて本発明は、付加的なハードウェア無しに、フォントキャッシュの一致を図る解決手段であり、装置コスト面で効果が大きい。

【００９５】個々の局所的なフォントキャッシュの内容がほぼ一致しているので、どの様なプロセスの割り当てが行なわれても、ある水準以上のキャッシュのヒット率を確保できるという効果がある。

【００９６】また、局所的なフォントキャッシュに該当する文字データが無い場合であっても、まず大域的なフォントキャッシュの管理テーブルの参照が行なわれるため、他のプロセスが既に作成し登録した文字データが有れば使用可能である。これは、同一文字のアウトラインフォント元データを複数のプロセスがハードディスク装置から読み取ろうとする処理の重複を防ぐ効果があり、結果的に、Ｉ／Ｏ処理がボトルネックとなることを解消する効果がある。

【図面の簡単な説明】

【図１】本発明の実施例のブロック図。

【図２】フォントキャッシュシステムの説明図。

【図３】マルチプロセッサ処理の説明図。

【図４】マルチプロセッサ処理の状態遷移図。

【図５】文字データ取得処理の概略構成図。

【図６】フォント管理テーブル内の一要素のデータ構造説明図。

【図７】カーネル処理の流れ図。

【図８】文字データ取得処理の流れ図。

【図９】文字データ登録処理の流れ図。

【図１０】文字データ削除処理の流れ図。

【図１１】読み出し処理の流れ図。

【図１２】書き込み処理の流れ図。

【図１３】連結解消処理の流れ図。

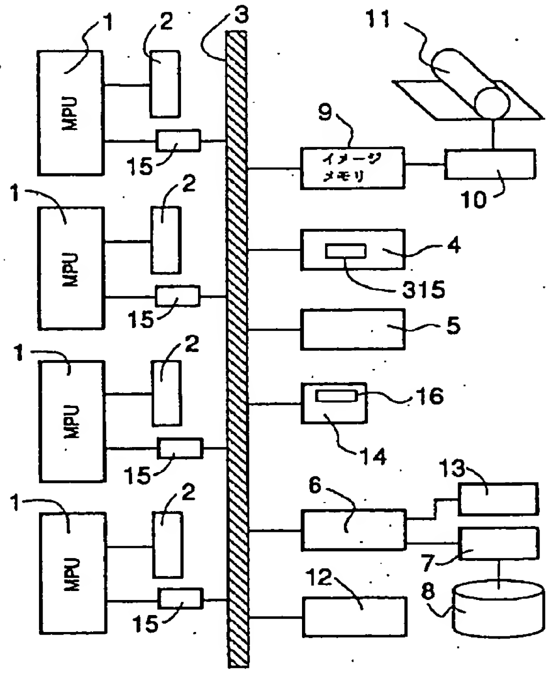
【図１４】更新処理304の別実施例の流れ図。

【符号の説明】

１…マイクロプロセッサユニット
２…フォントキャッシュ
３…グローバルバス
４…共有ＲＡＭ
５…共有ＲＯＭ
６…Ｉ／Ｏプロセッサ
７…ディスク制御装置
８…ハードディスク装置
９…イメージメモリ
１０…エンジン制御装置
１１…プリントエンジン
１２…バス調停制御回路
１３…インターフェース
１４…割り込み制御回路
１５…バス調停回路
１６…マイクロプロセッサユニット1の動作状態を示すフラグ
２０１…フォントデータ部
２０２…管理単位メモリ
２０３…指標部
２０４…データ部
２０５…あふれ領域
２０６…フォント管理テーブル
２０７…登録フラグ
２０８…文字指定レコード
２０９…キーの値（指標）
２１０…読み出し処理
２１１…書き込み処理
２１２…削除処理
３００…メモリ空間
３０１…待機ルーチン
３０２…ディスパッチルーチン
３０３、３０８、３０９…プロセス
３０４…更新処理
３０５…要求発生処理
３０６…カーネル処理
３０７…Ｉ／Ｏ処理
３１５…共有フォントキャッシュ領域
５０１…文字データ取得処理
５０２…文字データ登録処理
５０３…連結解消処理

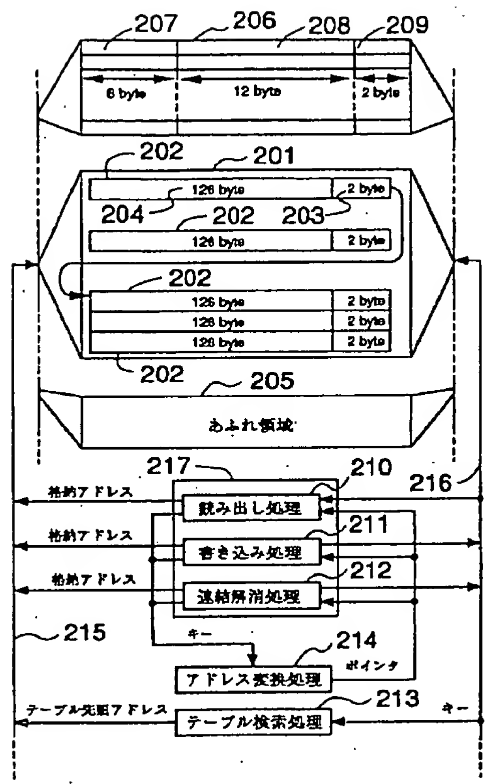
【図1】

1：マイクロプロセッサユニット
2：フォントキャッシュ
3：グローバルバス
4：共有RAM
5：共有ROM
6：I／Oプロセッサ
7：ディスク制御装置
8：ハードディスク装置
9：イメージメモリ
10：エンジン制御装置
11：プリントエンジン
12：バス調停制御回路
14：割り込み制御回路

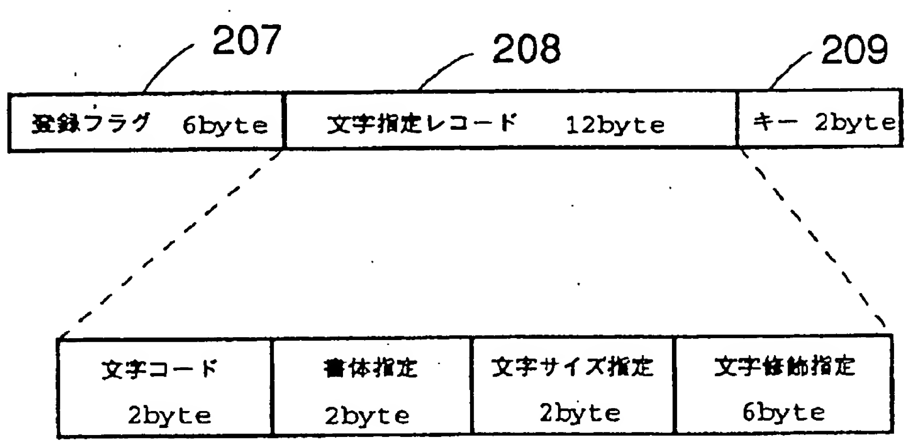


【図2】

215：アドレスバス
216：データバス

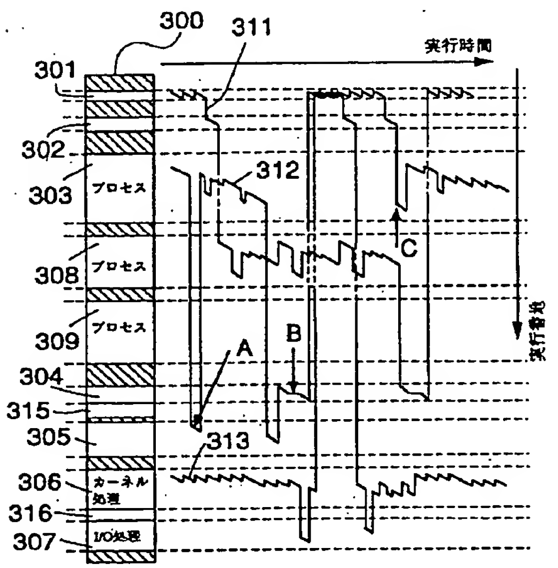


【図6】

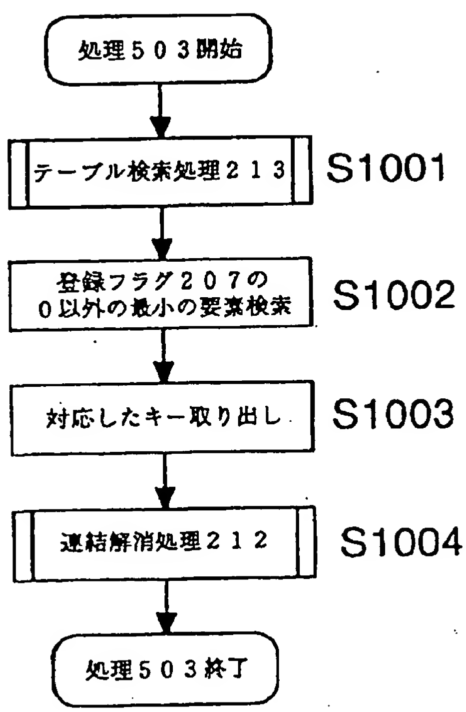


【図３】

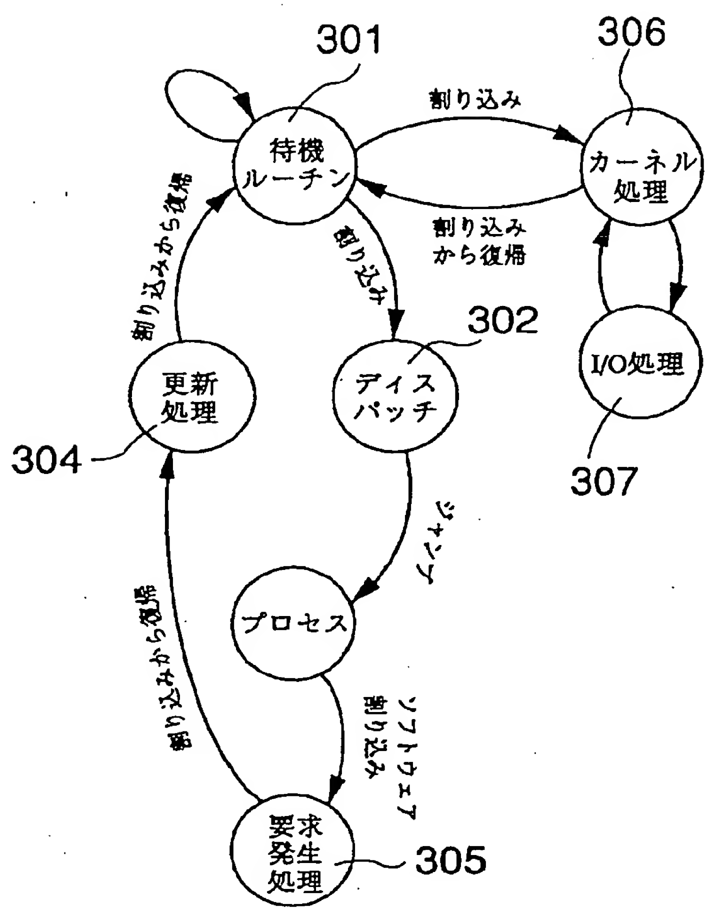
３００：メモリ空間
３０１：待機ルーチン
３０２：ディスパッチルーチン
３０３：プロセス
３０４：更新処理
３０５：要求発生処理
３０６：カーネル処理
３０７：Ｉ／Ｏ処理



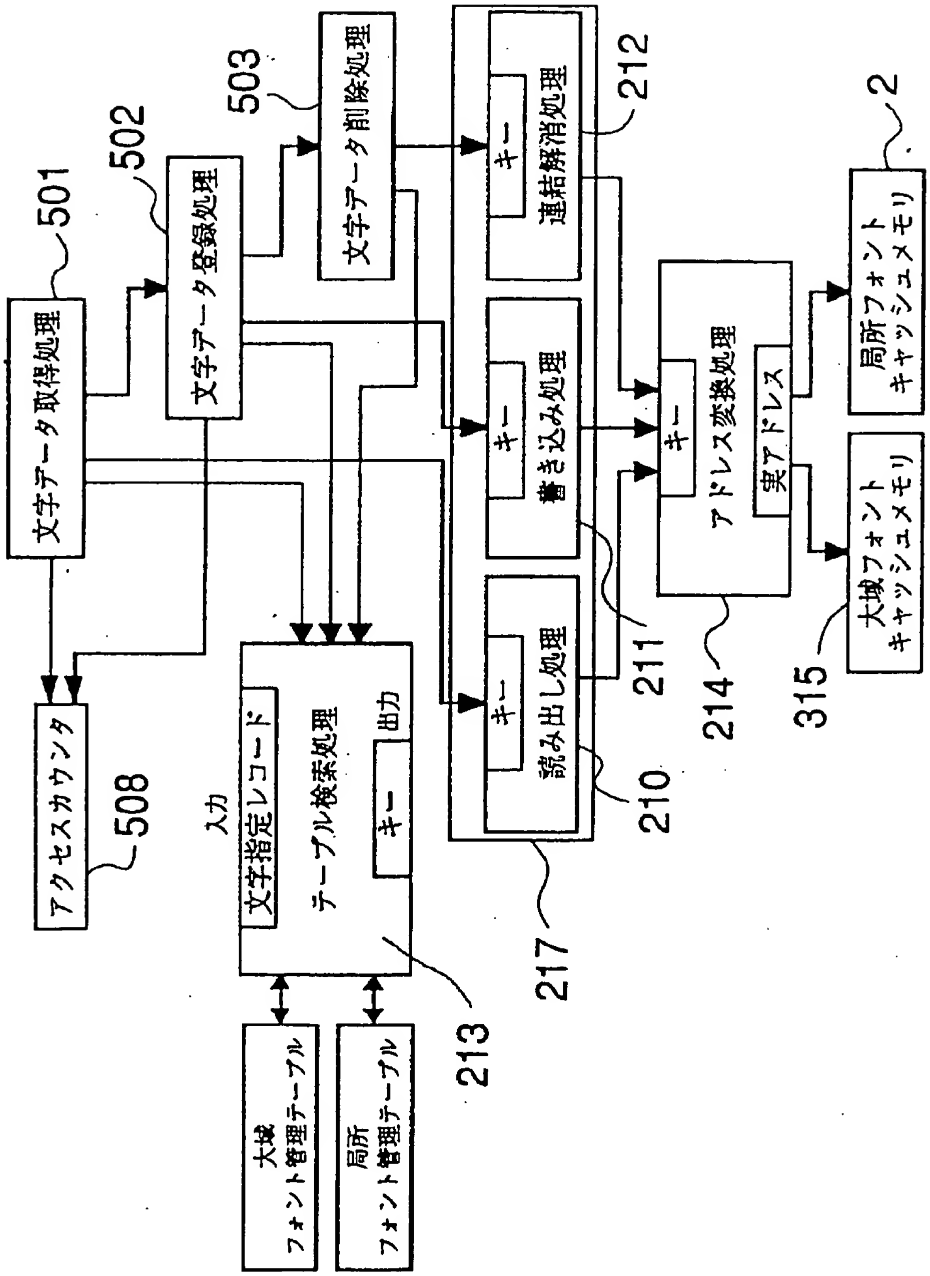
【図１０】



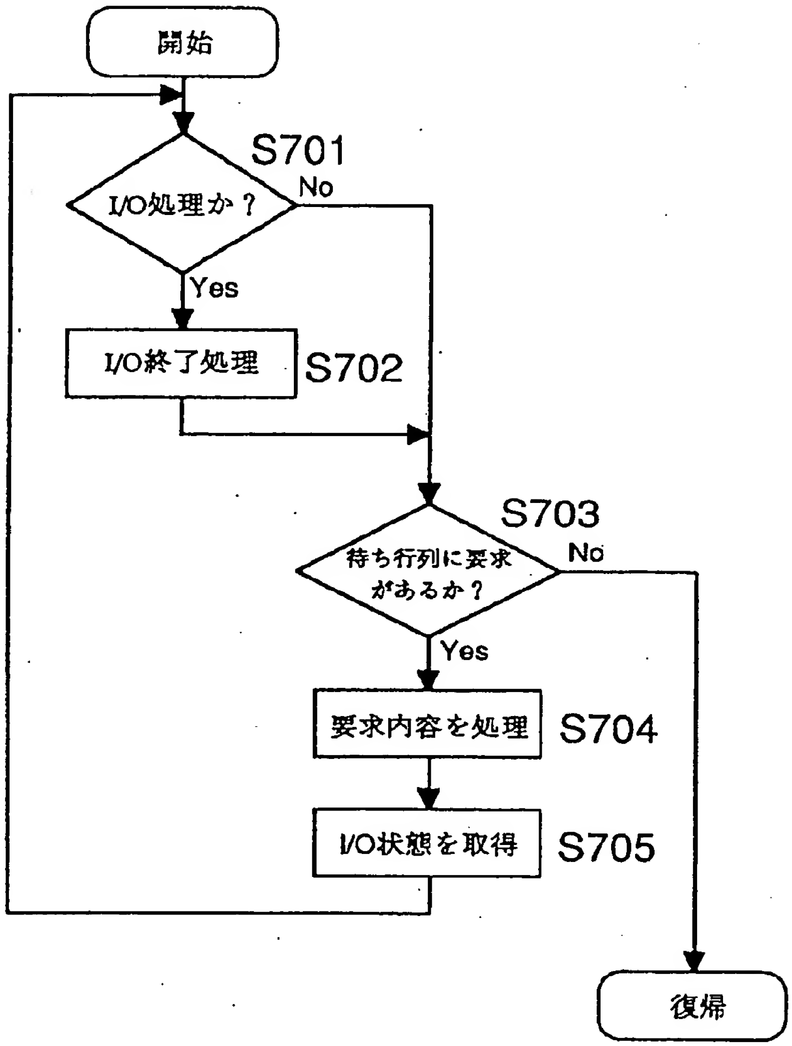
【図4】


301
306
待機
ルーチン
割り込み
カーネル
処理
割り込み
から復帰
割り込みから復帰
割り込み
302
ディス
パッチ
I/O処理
更新
処理
304
307
ジャンプ
プロセス
割り込みから復帰
ソフトウェア
割り込み
要求
発生
処理
305

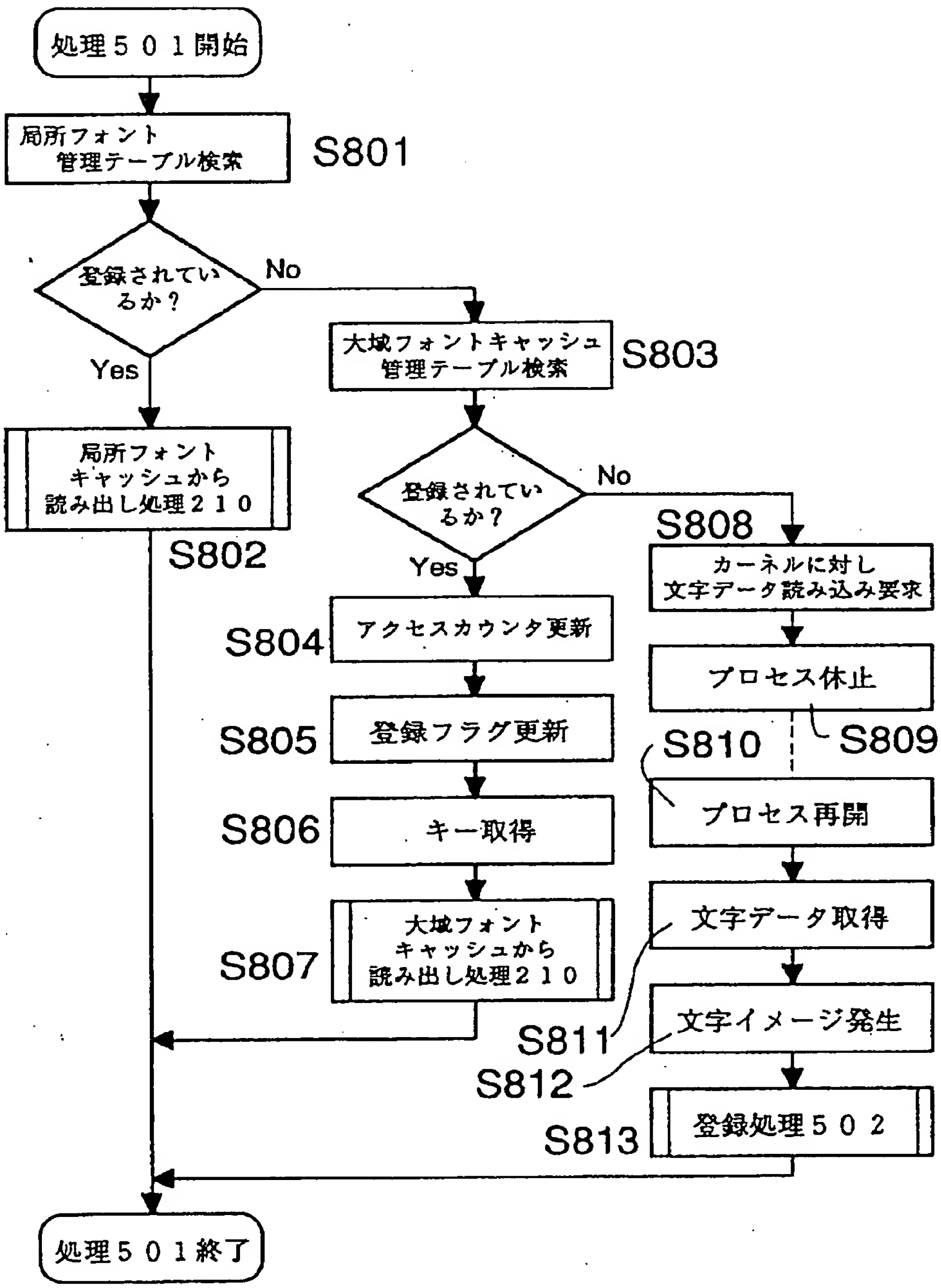
【図5】


文字データ取得処理
501
文字データ登録処理
502
文字データ削除処理
503
キー
連結解消処理
212
キー
書き込み処理
211
キー
読み出し処理
210
217
キー
アドレス変換処理
実アドレス
214
局所フォント
キャッシュメモリ
2
大域フォント
キャッシュメモリ
315
アクセスカウンタ
508
入力
文字指定レコード
テーブル検索処理
出力
キー
213
大域
フォント管理テーブル
局所
フォント管理テーブル

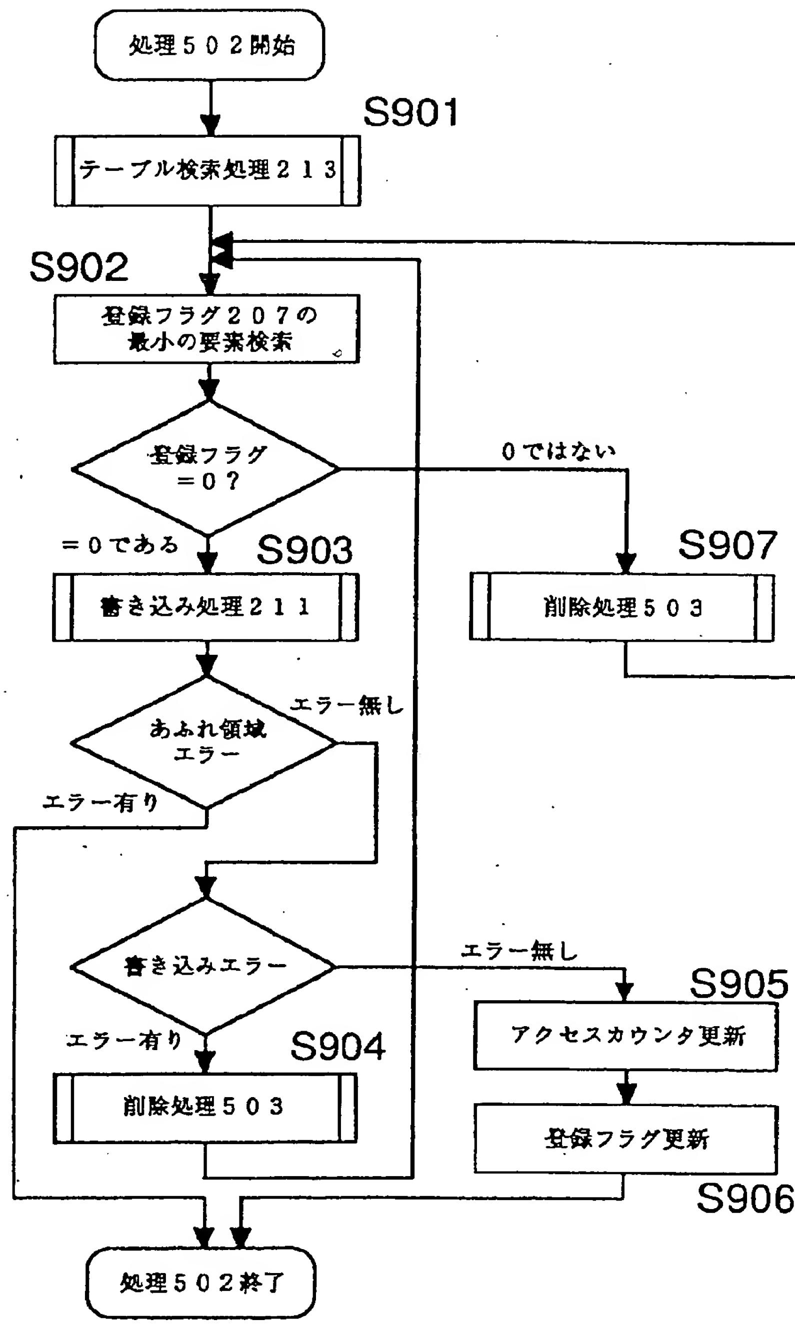
【図７】


開始
S701
I/O処理か？
No
Yes
I/O終了処理
S702
S703
待ち行列に要求があるか？
No
Yes
要求内容を処理
S704
I/O状態を取得
S705
復帰

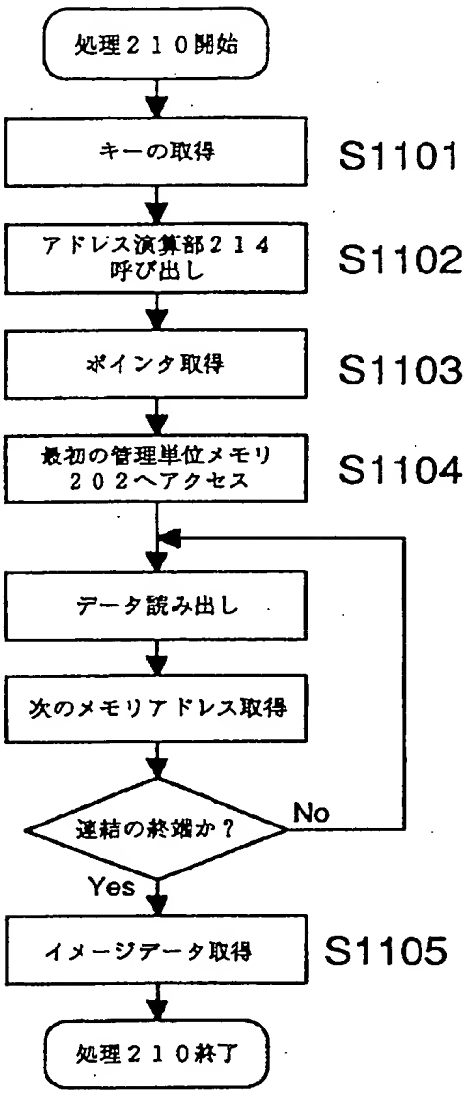
【図8】


処理５０１開始
局所フォント管理テーブル検索
S801
登録されているか？
No
Yes
局所フォントキャッシュから読み出し処理２１０
S802
大域フォントキャッシュ管理テーブル検索
S803
登録されているか？
No
Yes
S808
カーネルに対し文字データ読み込み要求
アクセスカウンタ更新
S804
プロセス休止
S809
登録フラグ更新
S805
S810
プロセス再開
キー取得
S806
文字データ取得
大域フォントキャッシュから読み出し処理２１０
S807
文字イメージ発生
S811
S812
登録処理５０２
S813
処理５０１終了

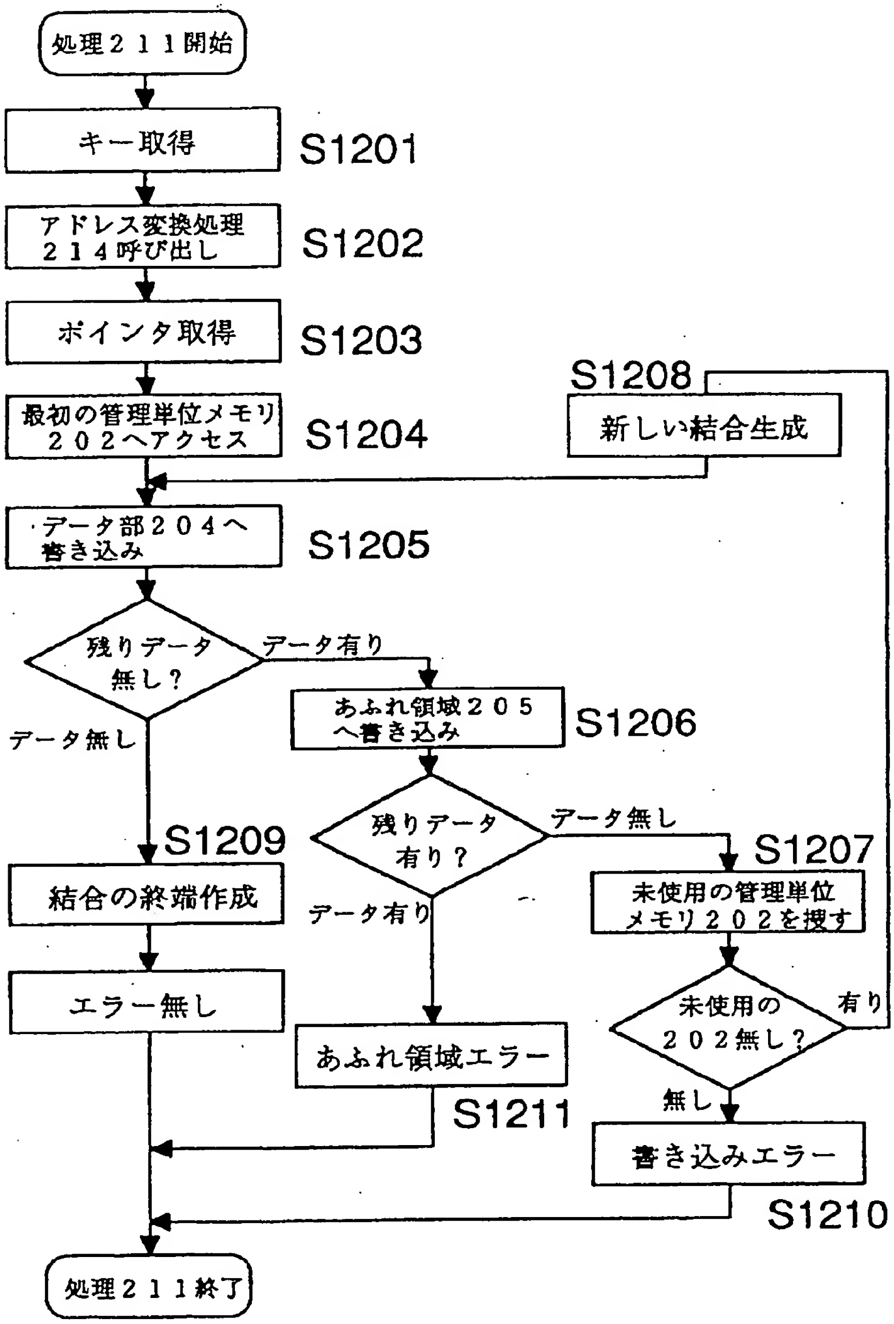
【図９】


処理５０２開始
S901
テーブル検索処理２１３
S902
登録フラグ２０７の
最小の要素検索
登録フラグ
＝０？
０ではない
＝０である
S903
書き込み処理２１１
S907
削除処理５０３
あふれ領域
エラー
エラー無し
エラー有り
書き込みエラー
エラー無し
S905
アクセスカウンタ更新
エラー有り
S904
削除処理５０３
登録フラグ更新
S906
処理５０２終了

【図１１】

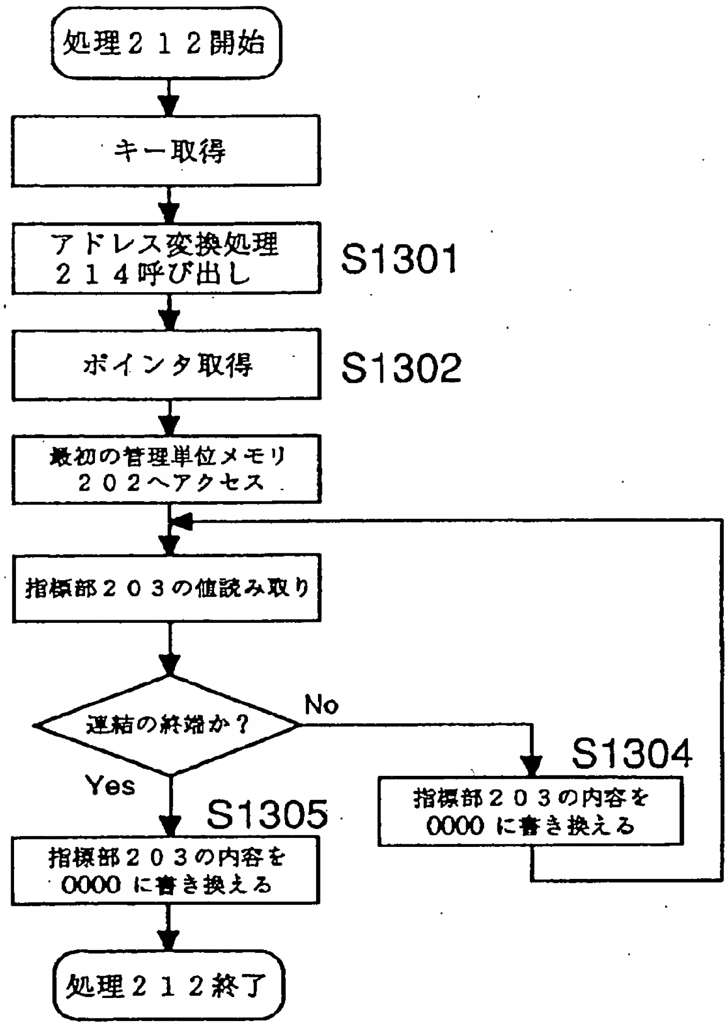

処理２１０開始
キーの取得
S1101
アドレス演算部２１４
呼び出し
S1102
ポインタ取得
S1103
最初の管理単位メモリ
２０２へアクセス
S1104
データ読み出し
次のメモリアドレス取得
連結の終端か？
No
Yes
イメージデータ取得
S1105
処理２１０終了

【図１２】

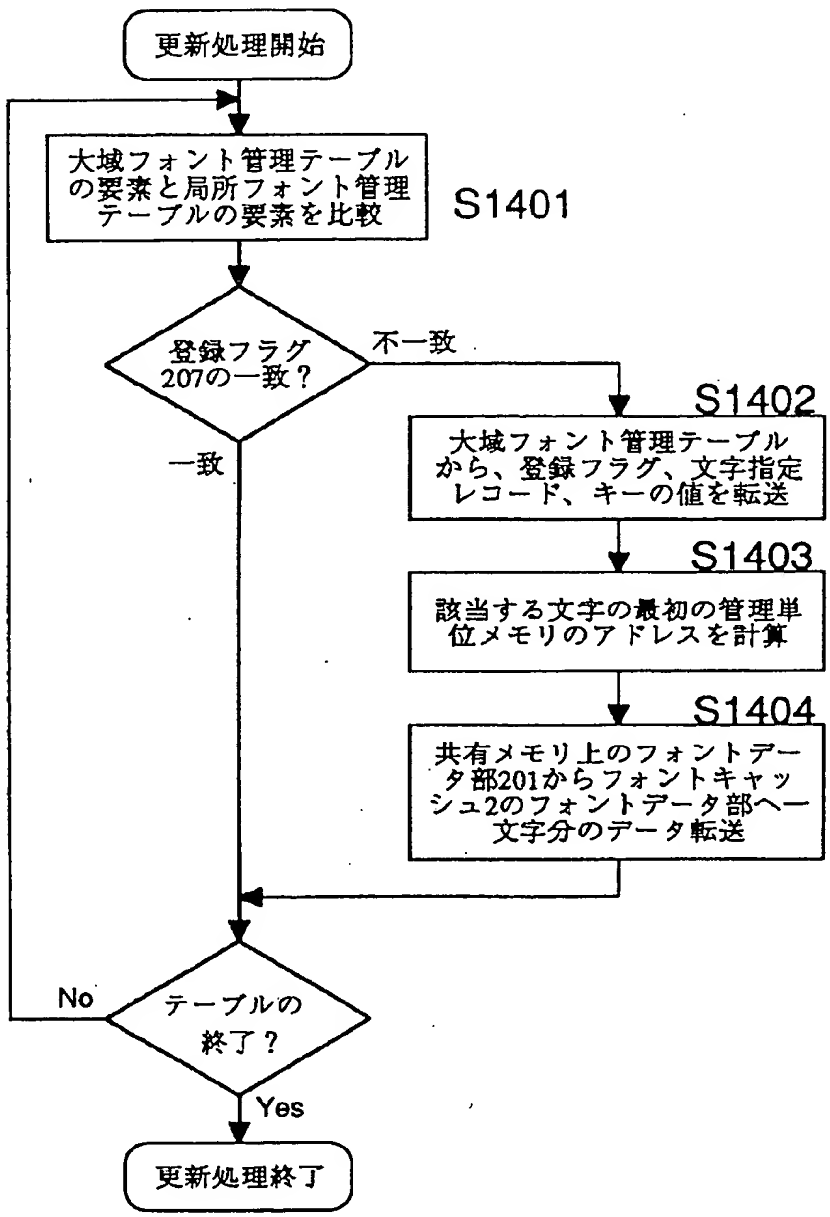


処理２１１開始
キー取得
S1201
アドレス変換処理２１４呼び出し
S1202
ポインタ取得
S1203
最初の管理単位メモリ２０２へアクセス
S1204
S1208
新しい結合生成
データ部２０４へ書き込み
S1205
残りデータ無し？
データ有り
あふれ領域２０５へ書き込み
S1206
データ無し
S1209
残りデータ有り？
データ無し
S1207
結合の終端作成
データ有り
未使用の管理単位メモリ２０２を捜す
エラー無し
未使用の２０２無し？
有り
あふれ領域エラー
S1211
無し
書き込みエラー
S1210
処理２１１終了

【図13】


処理２１２開始
キー取得
アドレス変換処理
２１４呼び出し
S1301
ポインタ取得
S1302
最初の管理単位メモリ
２０２へアクセス
指標部２０３の値読み取り
連結の終端か？
No
S1304
指標部２０３の内容を
0000 に書き換える
Yes
S1305
指標部２０３の内容を
0000 に書き換える
処理２１２終了

【図14】



フロントページの続き

(51)Int.Cl.^{5}		識別記号	庁内整理番号	F I	技術表示箇所
G09G	5/22		9061－5G		